Understanding Tables on the Web

Jingjing Wang[1]*, Haixun Wang[2], Zhongyuan Wang[2], and Kenny Q. Zhu[3]**

[1] University of Washington
[2] Microsoft Research Asia
[3] Shanghai Jiao Tong University

Abstract. The Web contains a wealth of information, and a key challenge is to make this information machine processable. In this paper, we study how to "understand" HTML tables on the Web, which is one step further from finding the schemas of tables. From 0.3 billion Web documents, we obtain 1.95 billion tables, and 0.5-1% of these contain information of various entities and their properties. We argue that in order for computers to understand these tables, computers must first have a brain – a general purpose knowledge taxonomy that is comprehensive enough to cover the concepts (of worldly facts) in a human mind. Second, we argue that the process of understanding a table is the process of finding the right position for the table in the knowledge taxonomy. Once a table is associated with a concept in the knowledge taxonomy, it will be automatically linked to all other tables that are associated with the same concept, as well as tables associated with concepts related to this concept. In other words, understanding occurs when computers will understand the semantics of the tables through the interconnections of concepts in the knowledge base. In this paper, we illustrate a two phase process. Our experimental results show that the approach is feasible and it may benefit many useful applications such as web search.

1 Introduction

The World Wide Web contains a wealth of information. Unfortunately, most of this information is understood only by humans but not by machines. A key challenge is thus to make such information machine accessible and processable.

In this paper, we focus on enabling machines to understand information on the Web. We are particularly interested in HTML tables, for the following two reasons. First, there are billions of tables on the Web, and many of them contain valuable information. Second, tables are well structured and relatively easier to understand, whereas converting text into structured data using natural language processing techniques is very costly and impractical for large web corpus.

Informally, we define the problem of *understanding* a web table as the problem of associating the table with one or more *semantic concepts* in a general purpose knowledge base we have created. In particular, after the association is established, each row of the table will describe the attributes of a particular *entity* of that concept (In this work, we assume tables are horizontal). For example, consider the tables in Figure 1. Our goal is to associate, Table (a) to such concepts as *US presidents* in the knowledge base. This

* The work was done at Microsoft Research Asia.

** Kenny Q. Zhu was partially supported by NSFC grants No. 61033002 and 61100050.

(a)

Barack Obama	Aug 4, 1961	Illinois
George W. Bush	July 6, 1946	Texas

(b)

Illinois	Springfield	12,900,000
Texas	Austin	25,145,561

(c)

Name	**Political Party**
George W. Bush	Republican
Hillary Clinton	Democratic

Fig. 1. Three Web Tables

means the knowledge base contains *US presidents* as a concept, as well as at least some of its instances (e.g., *George W. Bush*) and some of its attributes or properties (e.g., *date of birth*). Similarly, we need to associate Table (b) to such concepts as *US states*, which contain attributes such as *capitals* and *populations* and Table (c) to such concepts as *US politicians*, which contain attributes such as *political parties*. Note that the knowledge base may not contain all the information in the table including instances, attributes or values, but the knowledge base need to contain enough information for the machines to "guess" what the table is about. Note that this is different from schema detection. For example, in Table (c), the schema is given, but without associating the table to concepts such as *US politicians*, or *politicians*, it is not possible to understand the meaning of the data.

Once we understand tables, information contained in them can be used to support applications such as semantic search. The following examples show some "smart" answers to semantic queries (assuming Figure 1 contains all the tables for input).

If a user asks for "*Obama birthday*," the system returns the following table (For this work, we use tables to answer any queries. Of course, answers can be provided in any form, including in natural languages):

President	**Date of Birth**
Barack Obama	Aug 4, 1961

This shows that the system correctly identified the first column of Table(a) is about *presidents* and the second column about their birthdays.

Second, for query "*Illinois*," it returns the following table about the state of Illinois:

State	**Capital City**	**Population**
Illinois	Springfield	12,900,000

Note that i) the result has an inferred schema (a header), and ii) even though the word *Illinois* appears in Table (a) as well, our system realizes Table (a) is more about presidents than states, and the query is looking for information about Illinois, the state.

Finally, when asked for "*politicians*", the machine returns the following table by combining Table (a) and (b) and adding a header:

Politician	**Date of Birth**	**State**	**Political Party**
Barack Obama	Aug 4, 1961	Illinois	-
George W. Bush	July 6, 1946	Texas	Republican
Hillary Clinton	Oct 26, 1947	-	Democratic

What is interesting about this answer is that the system not only knows that Table (a) is about US presidents and Table (b) is about politicians, but it also knows that presidents are politicians, the two tables can be combined for the query. Such semantic association and reasoning between tables is significantly more intelligent than relational schema induction techniques which merely generate a "data type" for each column of a table.

Knowing the structure of the data does not mean knowing the semantics of the data, or understanding its content. For example, given Figure 1(a), how do we know *Barack Obama* is a president, and *Aug 4, 1961* is not just a date, but also Obama's birth date? Furthermore, how do we know that the first row is about Obama, the president, and not Illinois, the state? And finally, how do we know we can merge Figure 1(a) and (c)?

As human beings, to answer these questions, we need certain background knowledge. If we know Obama and Bush are US presidents, then immediately we can add title "*President*" to the first column of Figure 1(a). Next, we recognize that the second column is dates. Since our knowledge tells us presidents are persons and one of the most important dates about a person is his or her date of birth, we can add the "*Date of Birth*" header to the second column. If our knowledge also knows that Illinois and Texas are both US states, then it is not difficult to tag the third column with "*State*". Now, given that this table has three column titles: *President*, *Date of Birth* and *State*, our common sense tells us that every row of this table is more about a president than about a state, because a president has a birth date and a home state as two of his important attributes, whereas a state does not have a date of birth and a president for sure!

We can see that the key for understanding the tables in the above example is to answer these two related questions:

1. What is the most likely concept that contains a set of given entities?

2. What is the most likely concept that has a set of given attributes?

We call the process to answer these two questions *conceptualization*. In this paper, we utilize a general-purpose taxonomy called Probase [1–4] as our knowledge to conceptualize the information in web tables and achieve understanding. Probase is made up of worldly facts automatically constructed from 50 Terabytes Web corpus and other data. One dramatic difference between Probase and any manually built taxonomy (e.g. Open Mind Common Sense [5], Freebase [6]) is that Probase has over 2.7 million concepts, which cover most if not all concepts about worldly facts in human minds. Each concept contains a set of entities ranked by their popularity or other scores, and also a set of attributes used to describe entities in that concept. Given the rich concepts and probabilistic scores of Probase, we have a better chance to understand web tables than using other manually built taxonomies, since web tables also contain diverse and probabilistic information, which is hard to be well captured manually.

Figure 2 shows a snippet of the Probase taxonomy. Oval boxes denote concepts and their attributes, rectangles denote entities, while arrows denote *isA* relations. More details about Probase is given in Section 2.

Figure 3 illustrates the overall process for understanding web tables, which is detailed in Section 3. We first try to detect the header in a table (Section 3.2). If no header can be detected, we generate one using the table content (Section 3.3). Then we identify the column that contains entities (others are attribute columns) (Section 3.4). If we fail to derive the header or the entity column, or the result quality is considered too low, we discard the table. Otherwise, the recognized content is *attached* to the Probase concept that best describes the entity column. This essentially *extends* the Probase. We

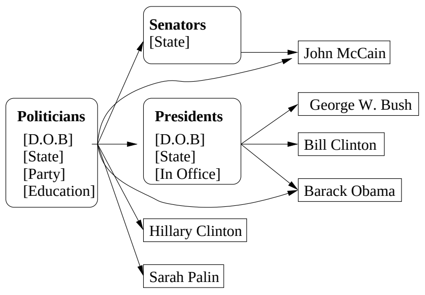

Fig. 2. A Snippet of the Probase Taxonomy

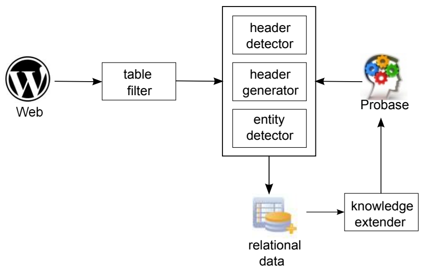

Fig. 3. The flowchart for understanding tables

evaluate our prototype system in Section 4, and discuss some of the most related work in Section 5 before concluding in Section 6.

2 Building a Knowledge Taxonomy

To conceptualize any web table, it is essential for our knowledge to contain large amount of concepts, their attributes and their representative entities. In this paper, we use Probase [1] to understand tables. Probase is a research prototype that aims at building a unified taxonomy of worldly facts from web data and search logs. The backbone of Probase is constructed by the Hearst patterns [7], or "SUCH AS" like patterns. For example, a sentence containing "... politicians such as Barack Obama ..." can be considered as an evidence that *politicians* is a hypernym of *Barack Obama*.

Despite their popularity in concept-entity extraction, Hearst patterns are not powerful enough for extracting attributes and values. As a result, the core Probase taxonomy is not rich in the attribute/value space. To enrich Probase with more attributes, we use the following linguistic pattern to discover seed attributes for a concept **C**:

What is the A of I?

Here, A is the seed attributes we want to discover, and I is an entity in concept **C** obtained from Probase. For example, assume we want to find seed attributes for concept *countries*. From Probase, we know *China* is a country. Then, we use the pattern "What is the * of China?" to search the Web. A match "What is the capital of China?" indicates *capital* is a candidate seed attribute for *countries*. The seed attributes thus extracted are good enough to detect many table schemas. Below, we address several important issues in attributes discovery. The notations used in our discussion are summarized in Table 1.

Table 1. Probase Notations

$\mathcal{C}, \mathcal{I}, \mathcal{A}$ concepts, instances, and attributes in the taxonomy

$E(c)$ entities of concept c	$p_c(i)$ plausibility of entity i for class c
$A(c)$ attributes of concept c	$a_c(i)$ ambiguity of entity i for class c
$C(i)$ concepts entity i belongs to	

1. Why use the rigid pattern "What is the A of I"? We deliberately choose a rigid pattern to increase the precision. Because i) we are only interested in seed attributes in this phase; ii) we use multiple entities to find attributes for each concept, and iii) we are matching the pattern against a huge Web corpus (50 Terabytes), which means we can always find good candidate attributes. Note that the above pattern is a skeleton which represents a set of concrete patterns. For example, the word *What* can be replaced by *Where* or *Who*, and *is* can be replaced by *are* or *was*.

2. What entities should we use as the **I** *in the pattern?* For each concept, Probase returns a list of entities associated with a set of scores. Two essential scores are *plausibility* and *ambiguity*. Plausibility measures how likely an entity is really a member of a given concept, and ambiguity measures how likely an entity is also a member of other unrelated concepts. For example, both *Microsoft* and *Apple* are companies, but *Apple* is more ambiguous because it is also a fruit. If we use *Apple*, we may mistake *taste* as an attribute of *companies*. So we want entities with high plausibility and low ambiguity. For how the scores are derived, we refer readers to [1]. Specifically, we find $ES(c)$, the eligible entities of class c:

$$ES(c) = \{i | i \in E(c), p_c(i) > \delta_p, a_c(i) < \delta_a\},$$

where δ_p and δ_a denote the thresholds for plausibility and ambiguity, respectively.

3. How to rank candidate seed attributes to obtain final seeds? For each concept c, we score and merge candidate seed attributes derived from $ES(c)$ to obtain final ones. If attribute a is derived with entity i, then we add a weight equals to $p_c(i)$ to a. We then aggregate the weights of all the candidate seed attributes, and choose top-ranked attributes as the final seed attributes of c. Specifically, let $D(a)$ denote the set of entities in $ES(c)$ having attribute a. The weight of a candidate attribute a with respect to c is:

$$w_c(a) = \sum_{i \in D(a)} p_c(i) / \sum_{i \in ES(c)} p_c(i). \tag{1}$$

Note that Eq (1) considers both plausibility and ambiguity, because $i \in ES(c)$.

One problem of using linguistic patterns to find attributes from the Web is noises, or wrong attributes. However, for different entities I, the pattern typically returns different noises, hence the ranking scheme effectively removes such noises.

Besides the above pattern, we also get attributes from DBPedia. We use exact matchings when locating Probase's entities in it.

In total, we obtain 10.5 million raw seed attributes (0.21 million distinct) for about 1 million classes. These are enough to bootstrap the process of understanding tables. Once we identified table schema, they can be used to expand the current set of attributes and essentially enrich Probase. In an evaluation of 30 concepts and their top 20 seed attributes, we found our approach reaches an average precision of 0.96. This provides a strong foundation for further understanding process.

3 Understanding Tables

From 0.3 billion web documents, we extract 1.95 billion raw HTML tables. Many of them do not contain useful or relational information (e.g., be used for page layout purpose); others have structures that are too complicated for machines to understand. We use a rule-based filtering method to acquire 65.5 million tables (3.4% of all the raw tables) that contain potentially useful information.

In this section, we describe the process of *understanding* a table. With the help of "enriched" Probase, we identify entities (of a single concept/class), attributes, and values in a table. Then, we use the information in the table to enrich Probase.

3.1 Knowledge APIs for Schema Extraction

We first introduce two functions $\kappa_A(A)$ and $\kappa_E(E)$, which are part of the knowledge API provided by Probase. Probase contains rich information about concepts, entities, and attributes. For a given concept, we may use Probase to find its entities and its attributes. Likewise, for a given set of entities or attributes, we may use Probase to find the set of concepts that they may belong to. The two functions serve this purpose. Given a set of attributes A and a set of entities E, functions $\kappa_A(A)$ and $\kappa_E(E)$ find the most likely concept to which A or E belongs [4]. More specifically, we have

- $\kappa_A(A)$: for a set of attributes A, $\kappa_A(A)$ returns a list of triples $\cdots, (c_i, A_i, sa_i), \cdots$ ordered by score sa_i, where c_i is a likely concept for A, $A_i \subseteq A$ are attributes of concept c_i, and sa_i is the score indicating the confidence of c_i given A.
- $\kappa_E(E)$: for a set of entities E, $\kappa_E(E)$ returns a list of triples $\cdots, (c_i, E_i, se_i), \cdots$ ordered by score se_i, where c_i is a likely concept for E, $E_i \subseteq E$ are entities of concept c_i, and se_i is the score indicating the confidence of c_i given E.

Table 2. A running example for table understanding

Name	**Birthdate**	**Political Party**	**Assumed Office**	**Height**
Barack Obama	4 Aug 1961	Democratic	2009	6'1
Arnold Schwarzenegger	30 Jul 1947	Republican	2003	6'2
Hillary Clinton	26 Oct 1947	Democratic	2009	5'8

The two functions may help us discover the schema of a table. Let us take Table 2 as an example. Suppose we want to know if the first row is the head of the table. As we know, a head usually contains a set of attributes of a concept. So by applying $\kappa_A()$ on $A = \{$*Name, Birthdate, Political Party, Assumed Office, Height*$\}$, we may expect to get some concepts of high confidence. Assume Probase returns the following for $\kappa_A(A)$:

(*US presidents, {Birthdate, Political Party, Assumed Office}, 0.90*)
(*politicians, {Birthdate, Political Party, Assumed Office}, 0.88*)
(*NBA players, {Birthdate, Height}, 0.65*)
...

The result ranks *US presidents* higher than *politicians*. This is probable because we are only looking at the header, and have not studied the table content yet.

On the other hand, if we compute $\kappa_A(A)$ on the second row $A = \{$*Barack Obama, 4 Aug 1961, Democratic, 2009, 6'1*$\}$, we may get empty results or results with very low confidence. This indicates that $\kappa_A()$ is useful in table header detection. We discuss this in more detail in Section 3.2.

However, many tables do not have headers. Function $\kappa_E()$ is useful in this case. Consider the first column E = $\{$*Name, Barack Obama, Arnold Schwarzenegger, Hillary Clinton*$\}$. Applying $\kappa_E()$ on E may get:

[4] The implementation of the two functions is discussed in [1, 8].

(*politicians, {Barack Obama, Arnold Schwarzenegger, Hillary Clinton}, 0.95}*)
(*actors, {Arnold Schwarzenegger}, 0.5}*)
...

In this case, $\kappa_E()$ identifies *politicians* as the top match. If we apply $\kappa_E()$ on other columns, we may recover more possible attributes. Thus, there is a possibility we can generate a header for the table. We discuss this in more detail in Section 3.3.

In summary, combining the outcomes of $\kappa_A()$ and $\kappa_E()$, we may get a clue regarding what the table is about. The scores sa_i and se_i returned by $\kappa_A()$ and $\kappa_E()$ play an important role in reaching the conclusion. One important issue is how sa_i and se_i are calculated. Intuitively, the more matching attributes/entities we find, the higher score of a concept should be. Specifically, assuming (c_i, E_i, se_i) is in the output of $\kappa_E(E)$ and (c_i, A_i, sa_i) is in the output of $\kappa_A(A)$, a straightforward way to define se_i and sa_i is:

$$se_i = \frac{1}{|E|} \sum_{e_i \in E_i} p_{c_i}(e_i), \qquad sa_i = \sum_{a_i \in A_i} w_{c_i}(a_i) \tag{2}$$

where $p_{c_i}(e_i)$ is the plausibility score of e_i given c_i, and $w_{c_i}(a_i)$ is the confidence score of a_i being the attribute of c_i in Probase.

3.2 Header Detector

The first step in understanding a table is locating the header. The header of a table contains a set of attributes, which form the schema of the table. It also indicates the orientation, depending on whether it appears as a row or column. In our discussion, we assume all tables are horizontal, e.g. in Table 2, the header is the top row.

Let P denote the set of possible headers for a table T. In one extreme, we may consider every row of T, that is, P contains all rows of T. In the other extreme, we may just consider the top row, which is the mostly header if T has one. We use function $\kappa_A()$ to evaluate each possibility and generate a set of candidate schema.

$$candidate\_schema = \{x \mid p \in P, x \in \kappa_A(p), x.sa + \alpha(p, T) > \gamma_h\} \tag{3}$$

Eq 3 filters out candidates whose score is below a threshold γ_h.[5] The score consists of two parts, the raw sa score returned by $\kappa_A()$, and an adjustment $\alpha(p, T)$, which evaluates whether a row is likely a header based on the syntax. The reason we need an adjustment is that the header usually has some syntactic characteristics that set it apart from the rest of the table. To name a few:

- HTML <th> tag.
- Other syntactic clues, including i) cells of p end with a colon ':', and ii) cells of p has a formatting (e.g., **bold** font is used) different from other rows in table T, etc.
- Data type differences[6]: The fact that a cell in p is of different data type from other cells in its column may indicate p is a header. For example, a cell in p has string type (e.g., *Birthdate* or *Age*), while other cells in that column are dates (*4 Aug 1961*) or numbers (e.g., *61*).

[5] γ_h was picked based on our observation in practice.
[6] Although data types are not an entirely syntactic issue, we included it here since they can be detected syntactically.

If $candidate\_schema$ returned by Eq 3 is empty, then we fail to detect any header. This is possible because tables may not come with a header. In this case, we generate a header using the method elaborated in Section 3.3.

As for the example of Table 2, a properly set threshold will find the first row as the header, with two candidate schema that are shown below. For simplicity, we assume $\alpha(p, T) = 0$ here, that is, the table provides no syntactic clues for header identification.

(*US presidents, {Birthdate, Political Party, Assumed Office}, 0.90*)
(*politicians, {Birthdate, Political Party, Assumed Office}, 0.88*)

In Section 3.4 we will discuss how to further narrow the above two candidates down to a single interpretation.

3.3 Header Generator

If no candidate header is found in the header detection phase, then we assume that the table does not come with a header. In this case, we try to generate a header for the table.

For each column L_i, we find its most likely concept. For example, from the 3rd column of Table 2, we may derive *political parties*. Then, we check if the concepts of all columns together describe some other concepts. Specifically, for each column L_i, we first find its best matched top-k candidate concepts by calling $\kappa_E(L_i)$. We denote the top-k candidate concepts of L_i as $c^1(L_i), \cdots, c^k(L_i)$. Then we generate P, a set of possible headers by selecting one concept from each of the top-k candidates, that is

$$P = \{(a_1, \ldots, a_i, \ldots, a_n) | a_i \in \{c^1(L_i), \cdots, c^k(L_i)\}\}$$

where n is the number of columns. Finally, we derive $candidate\_schema$ using P and Eq 3. Note that in this case, $\alpha(p, T)$ is 0, as the table does not have syntactic clues.

If $candidate\_schema$ is again empty, which means the table does not come with a header and we cannot generate one for it, then we drop it from further consideration.

3.4 Entity Detector

The entity detector tries to accomplish two tasks. First, it detects the *entity column* of the table. For example, for Table 2, the entity column is the 1st column instead of the 3rd column, as the table is about politicians, not political parties. Second, we have derived a set of candidate schemata previously, and we need to narrow them down to a final interpretation. The method presented here solves the two problems at the same time.

We make our judgment based on two kinds of evidence. First, the entity column should contain entities of the same concept, and we can derive the confidence of a concept for a given column. Second, the header should contain attributes that describe entities in the entity column, that is, the candidate schema we have derived should match the concept of the entity.

For each $s \in candidate\_schema$ returned by Eq 3, we enumerate every column col and compute its confidence score. Given a column col, we define

- E^{col}: the set of all cells in col, except for the one in the header corresponds to s
- A^{col}: the set of all attributes in s, except for the one in the current column.

We then apply functions $\kappa_A()$ and $\kappa_E()$ on E^{col} and A^{col} to obtain their possible semantics. Specifically, we let SC_A to be the ordered list of (c_i, A_i^{col}, sa_i) returned by $\kappa_A(A^{col})$, and SC_E to be the ordered list of (c_i, E_i^{col}, se_i) returned by $\kappa_A(E^{col})$.

If col is the entity column, we should be able to derive a concept c from E^{col}, and c should also be strongly supported by A^{col}. So the possibility of col being the entity column relies on the confidence of the concepts derived from it.

We join SC_A and SC_E to find common concepts, and we record the corresponding score as a multiplication of sa_i and se_j.

$$h(s, col) = \max\{sa_i \cdot se_j \mid (c_i, A_i^{col}, sa_i) \in SC_A,\ (c_j, E_j^{col}, se_j) \in SC_E,\ c_i = c_j\}$$

Finally, the most possible interpretation and entity column are the ones that achieve the maximum score.

$$(final\ schema, entity\ column) = \operatorname*{argmax}_{s,col} h(s, col)$$

Using this approach, for the example in Table 2, we will reject the schema of *US presidents*, because although the concept *US presidents* has strong support from the attributes, it has low support from the column that contains the name Arnold Schwarzenegger. Thus, the final schema we find for the table is:

(*politicians, {Birthdate, Political Party, Assumed Office}, 0.88*)

4 Experimental Results

Since most of the evaluations are related to concepts, we randomly selected 30 concepts with high frequency as our benchmarks, which vary in domains and sizes. For experiments that need to be evaluated manually, we use a consistent scoring criterion for human judges: 1 for correct, 0.5 for partially correct, 0 for incorrect. In the following subsections, we will first present some statistics about the extracted table corpus, and then two applications that can take advantage of the web table understanding. Because of the space limitation, only some statistics of the results will be provided. Readers are referred to our website [7] for complete experimental results.

4.1 The Web Table Corpus

We implemented the prototype system with a Map-Reduce distributed computing platform. Since all of the following stages are parallelable, Each of them takes at most several hours to complete on our large cluster. We ask human judges to judge the correctness of the results.

1. Header detection: We randomly selected a sample of 200 [8] filtered tables, and ran the header detection algorithm on them. The result is shown below.

Actual no	86.7% correct, 13.3% incorrect
Actual yes	90.7% correct, 9.3% incorrect (2.1% wrong position, 7.1% predict as no header)

Our algorithm correctly identified headers (or lack of which) in 89.5% of the tables. Among the 140 tables with a header, we correctly detected the header in 127 (90.7%) of them.

[7] http://research.microsoft.com/en-us/projects/probase/tablesearch.aspx

[8] This number is limited since we have to use human judges

2. Entity column detection: Because the information in web tables is so diverse, random sampling from web tables may not have a good coverage in both structures and domains. Therefore in this part, we create test set using tables extracted from Wikipedia only. These tables contain structural multiple-domain data in high density. We only focus on precision here as recall is hard to evaluate. We randomly selected 200 tables which our system claims to have an entity column, while ensuring that no two tables are from the same Wikipedia page. Our judges evaluated the correctness of these claims. Results showed that 11 tables actually do not have an entity column, and most of them have two or more main entities, which violate our assumption that there is only one entity column in a table. Among the remaining 189 tables, our algorithm correctly identifies the entity column in 165 tables(87.3%).

4.2 Search Engine

We build a semantic search engine that operates upon table statements, rather than tables themselves. A statement consists of a single entity, its attributes and corresponding values. Usually, a statement corresponds to a row of a table. For example, Table 2 has three statements, each about a politician. A statement represents a basic unit of knowledge of an entity, which is used to answer a query. This is different from previous work [9], which uses tables as the basic unit to answer queries.

Furthermore, instead of performing keyword matching in table search as in previous work, we try to find the semantics of a query, and return a set of statements that match the semantics. We support four semantic components in a query: Concept, Entity, Attribute and Keyword. Different combinations of them lead to different types of query. In this experiment, we focus on one representative pattern of them: Concept + Attribute, because this pattern includes two main factors: the relation between entity and concept, and the relation between attribute and concept. Some examples of pattern include "politicians birthday", "companies industry" and "video games developer". Once we have the set of statements that match a query, we rank them according to two factors: 1) match of keywords and attributes between the query and the statement; 2) the quality of the statement.

To avoid indexing all the tables on the web, we implemented the search engine only on Wikipedia's tables. For the 30 concepts, we select three attributes from each of them and generate corresponding queries. Considering we need to have a certain number of results for each query to make the evaluation possible, we ask users to first think of attributes they want to query, and then choose at most three of them with enough results.

To evaluate both the precision and quality of ranking, we ask the users to score each of the at most top 10 results of each query. Let $score_k$ denote the score for the result at position k among n results, then the overall quality of a query result is defined as:

$$\frac{\sum_{k\in[1,n]} score_k/k}{\sum_{k\in[1,n]} 1/k}.$$

In total 82 out of 90 queries return enough results. Among them, 53 queries return all-correct results, and 6 queries have a score lower than 0.6. The average score of all queries is 0.91. Figure 4 shows the average scores of queries for each concept. [9] Among all the 29 concepts which have result, 25 of them received an score higher than 0.8.

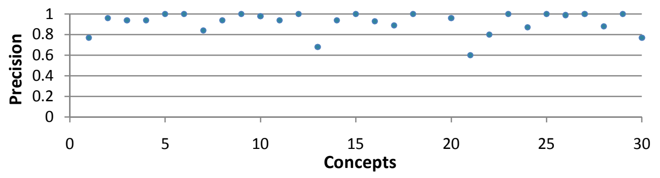

Fig. 4. Average score of queries for each concept

To compare with an existing search engine, we submitted the same queries to Google. For each query, we manually judge the top 10 pages returned to see if they contain the desired information which are forms, lists or categories about the queried class and attribute. Out of the 820 results (10 for each query), 32 results from Wikipedia and 81 results from non-wiki websites are considered useful. But the formats of these pages make it difficult if not impractical to extract the information we need. This experiment showed that traditional keyword search is inadequate in handling such semantic queries.

To better evaluate the quality of entities and values, we compare our results with Google Squared, which also outputs tables as results. Our prototype search engine merges statements with the same schema together as one table. We select at most top 10 schemata ranked by the highest score of their associated statements. For each query, we obtain a list of entities and their attribute values, then compare it with the list returned by Google Squared. We manually checked the correctness of our list and ensure that the entities and values are correct. Figure 5 shows the percentage of our result that is included by, overlapped with or disjoint with the list from Google Squared for all 30 concepts. We argue that our system not only correctly contains attribute-value information that Google Squared currently has for the 30 concepts, but also includes many additional attributes and values that are not found in Google Squared.

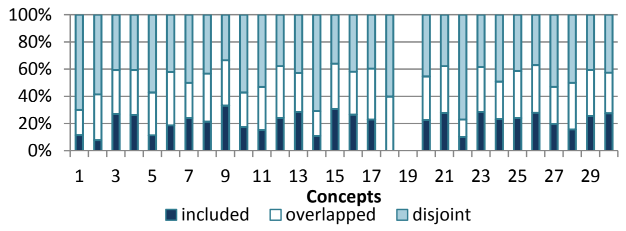

Fig. 5. Comparing with Google Squared

4.3 Taxonomy Expansion

In this experiment, we evaluate how much our prototype system contributes in expanding the Probase taxonomy in terms of the number of entities and attributes.

1. Entity expansion: Expanding entities is straightforward. For example, based on Probase, we conclude that the leftmost column in Table 2 contains names of politicians,

[9] some concepts do not have dots or bars in a chart because there aren't enough results for them (similarly hereinafter)

even if not every name there is found in Probase. The leftmost column provides an evidence equivalent to a Hearst's pattern "*... politicians such as Barack Obama, Arnold Schwarzenegger and Hillary Clinton ...*" Thus, we can expand Probase by incorporating this new evidence in the same way as we encounter a new Hearst's pattern.

To ensure a high quality, we select top 1000 entities ranked by ambiguity score $a_c(e)$ as seeds for each concept c, then use plausibility score $p_c(e)$ to infer. Here we set δ_p to 0.6. We run our algorithm for one iteration, and discovered 3.4 million existed entities in Probase, but more importantly, 4.6 million new entities for nearly 20,000 concepts. Among the 30 concepts, about 11,000 new entities were discovered from 28 of them, in which 1,410 (12.8%) are also found in Freebase by exact matching. Similar as before, we judge the top 20 entities (if available) manually for each concept, which are sorted by the maximal value of p_c, and use the average score to indicate a concept's quality.

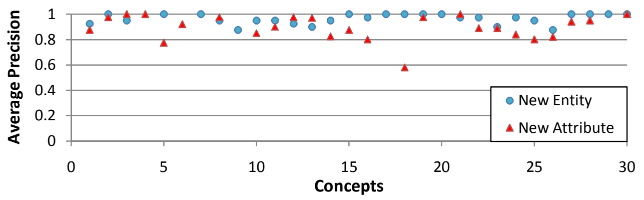

Fig. 6. Precision of new entities and new attributes

The result is shown in Fig 6. The average quality is 0.96, and all concepts have score higher than 0.875. Furthermore, new entities in 12 of the concepts are all correct. In essence, most of the newly discovered entities are correct.

2. Attribute expansion: After the entity column is determined, we can expand the attribute list in Probase by adding unknown attributes to representative concepts of the entity column. This process can help us get more attributes in the light of tables, such as "Tel-#" for "phone number". Furthermore, enriching Probase by expanding attribute set and understanding more tables are in positive feedback loop.

We again run our algorithm for one iteration, and discovered 0.15 million new attributes for nearly 14,000 concepts. About 2,700 new attributes were discovered for all the 30 concepts. Like before, we care more about the quality of top-ranked attributes than the general quality. For each concept, new attributes are first filtered though frequency and average $w_c(a)$, and then sorted according to the maximal $w_c(a)$. We still judge the top 20 attributes manually and use the average score to evaluate the concept. Here we set $\delta_a = 0.4$.

The result is shown in Fig 6. We found new attributes for all 25 concepts, and the average quality of them is 0.90. The only outlier is the concept "guitarists", whose quality score is 0.58. That's because "guitarists" are often associated with "songs" in tables. But since Probase's coverage of songs is not good, guitarists are selected as the entity column instead which is not the best choice.

5 Related Work

There has been some early work on information extraction from tables on the Web. Wang *et al.* [10] detects tables by using machine learning methods. It uses features of layout, content type and word usage for training and testing different classifiers.

However, the corpus only contained about 10,000 tables collected from 200 web sites. Yoshida *et al.* [11] studied the problem of integrating tables of similar content into a big table. It uses EM method to decide the locations of attributes and values in a table by labeling the table format as a predefined structure, then group tables into several clusters according to their attributes and values. Similar attributes in a cluster are merged together to get a final big table. Another early work [12] focused on mining web tables in a specific domain: travel. It defines similarity between cells, rows, and columns, then uses the similarity to identify the schema of the table.

Cafarella *et al.* [13, 9] did some pioneering work in exploring tables on the Web. The goal is to decide if a table contains *relational* data. To achieve this, they tried to identify typed columns in the table, which resulted in schemata for tables that are considered relational. They also studied an interesting application: searching over a corpus of tables. More recently, a related system called Octopus [14] explored the semantic matching among tables and achieved better accuracy and performance. Work in this space relied on correlation statistics inside the tables instead of external knowledge. Furthermore, each table is treated as a single entity (tables are atom units in query results) whereas we consider table as a set of entities and values. Syed *et al.* [15] used *Wikitology*, an ontology which combines some existing manually built knowledge systems such as DBPedia [16] and Freebase [6], to link cells in a table to Wikipedia entities. To resolve the ambiguity issue (a cell may match many Wikipedia entities), it finds the class or the domain of a column (or a row) through majority voting to achieve more accurate matching. Other researchers are interested in labeling cell text, column type and relation between columns(rows) [17, 18] using a large corpus such as YAGO, and use the information to assist queries about relation. However, as we described in Section 1, people need knowledge, including common sense and worldly facts, to understand a table, especially when the table is taken out of its context. Without such knowledge, it is often impossible to truly understand the table, or to be able to perform the type of semantic search as described in this paper.

The work presented in this paper focuses on *understanding* tables. To obtain knowledge required for this task, we build it upon an extension of Probase [1], a rich ontology automatically derived from the Web to produce a concept map for the human mind. In our extension, we find attributes for each concept (there are more than 2 million of them in Probase). Some recent work has focused on finding attributes for given classes and concepts. In particular, Pasca *et al.* [19] finds class attributes starting from a few *seed* (entity, attribute) pairs for each class. Furthermore, a ranking mechanism [20] has been developed to score attributes derived from seeds.

Besides tables, some work has also been done to explore other sources of structured data on the Web, for example using data extracted from lists to construct relational data [21] or to expand set of entities [22]. There is also research that transforms text to tables [23, 24]. In particular, Pyreddy *et al.* [23] use heuristic rules to tag table components, and Pinto *et al.* [24] use conditional random fields to locate boundaries, headers and rows of text table.

6 Conclusion

This paper presents a framework that attempts to harvest useful knowledge from the rich corpus of relational data on the Web: HTML tables. Through a multi-phase algorithm, and with the help of a universal probabilistic taxonomy called Probase, the framework is capable of understanding the entitles, attributes and values in many tables on the Web. With this knowledge, we built two interesting applications: a semantic table search engine which returns relevant tables from keyword queries, and a tool to further expand and enrich Probase. Our experiments indicate generally high performance in both table search results and taxonomy expansion. This showed that the proposed framework practically benefits knowledge discovery and semantic search.

References

1. Wu, W., Li, H., Wang, H., Zhu, K.Q.: Probase: A probabilistic taxonomy for text understanding. In: SIGMOD. (2012)
2. Lee, T., Wang, Z., Wang, H., Hwang, S.: Web scale taxonomy cleansing. In: VLDB. (2011)
3. Zhang, Z., Zhu, K.Q., Wang, H.: A system for extracting top-k lists from the web. In: KDD. (2012)
4. Liu, X., Song, Y., Liu, S., Wang, H.: Automatic taxonomy construction from keywords. In: KDD. (2012)
5. Singh, P., Lin, T., Mueller, E., Lim, G., Perkins, T., Li Zhu, W.: Open Mind Common Sense: Knowledge acquisition from the general public. On the Move to Meaningful Internet Systems: CoopIS, DOA, and ODBASE (2002) 1223–1237
6. Bollacker, K., Evans, C., Paritosh, P., Sturge, T., Taylor, J.: Freebase: a collaboratively created graph database for structuring human knowledge. In: SIGMOD. (2008)
7. Hearst, M.A.: Automatic acquisition of hyponyms from large text corpora. In: COLING. (1992) 539–545
8. Song, Y., Wang, H., Wang, Z., Li, H., Chen, W.: Short text conceptualization using a probabilistic knowledgebase. In: IJCAI. (2011)
9. Cafarella, M.J., Wu, E., Halevy, A., Zhang, Y., Wang, D.Z.: Webtables: Exploring the power of tables on the web. In: VLDB. (2008)
10. Wang, Y., Hu, J.: A machine learning based approach for table detection on the web. In: WWW. (2002)
11. Yoshida, M., Torisawa, K., Tsujii, J.: A method to integrate tables of the world wide web. In: International Workshop on Web Document Analysis. (2001)
12. Chen, H., Tsai, S., Tsai, J.: Mining tables from large scale html texts. In: ICCL. (2000)
13. Cafarella, M.J., Wu, E., Halevy, A., Zhang, Y., Wang, D.Z.: Uncovering the relational web. In: WebDB. (2008)
14. Yakout, M., Ganjam, K., Chakrabarti, K., Chaudhuri, S.: Infogather: entity augmentation and attribute discovery by holistic matching with web tables. In: SIGMOD. (2012)
15. Syed, Z., Finin, T., Mulwad, V., Joshi, A.: Exploiting a web of semantic data for interpreting tables. In: Proceedings of the Second Web Science Conference. (2010)
16. Auer, S., Bizer, C., Kobilarov, G., Lehmann, J., Cyganiak, R., Ives, Z.: Dbpedia: A nucleus for a web of open data. In: ISWC/ASWC. (2007) 722–735
17. Limaye, G., Sarawagi, S., Chakrabarti, S.: Annotating and searching web tables using entities, types and relationships. In: VLDB. (2010)
18. Venetis, P., Halevy, A.Y., Madhavan, J., Pasca, M., Shen, W., Wu, F., Miao, G., Wu, C.: Recovering semantics of tables on the web. PVLDB **4** (2011)
19. Pasca, M.: Organizing and searching the world wide web of facts - step two: Harnessing the wisdom of the crowds. In: WWW. (2007)
20. Bellare, K., Talukdar, P.P., Kumaran, G., Pereira, F., Liberman, M., McCallum, A., Dredze, M.: Lightly-supervised attribute extraction. In: NIPS. (2007)
21. Elmeleegy, H., Madhavan, J., Halevy, A.: Harvesting relational tables from lists on the web. In: VLDB. (2009)
22. He, Y., Xin, D.: Seisa: set expansion by iterative similarity aggregation. In: WWW. (2011)
23. Pyreddy, P., Croft, W.B.: Tintin: A system for retrieval in text tables. In: ICDL. (1997)
24. Pinto, D., McCallum, A., Wei, X., Croft, W.B.: Table extraction using conditional random fields. In: SIGIR. (2003)