# Interactively Querying and Updating Freebase with Web Tables

**Alan Ritter** and **Evan Herbst**

## Introduction

Recently there have been a growing number of large general purpose databases, such as Freebase(Bollacker et al. 2008), Wikipedia, and DbPedia(Auer et al. 2007), which have been enabling many new applications. [1] These data sources rely on volunteers to manually enter structured data, which can be tedious, thus discouraging ordinary users from contributing. Some recent work has addressed this issue, by automatically extending these resources with information extraction techniques(Wu and Weld 2007), and providing an interface which allows casual users to quickly verify extracted results(Hoffman et al. 2009).

In addition to data extracted from unstructured text, there exists a large amount of semi-structured data on the web, much of it in the form of HTML tables (Cafarella et al. 2008). This data could be harvested to automatically extend databases such as Freebase, however there are many data integration challenges. It seems unlikely that 100% accuracy can be achieved, which is unacceptable for many applications, so it is necessary to rely on humans to map these data sources to Freebase. Ordinary users, however, are unlikely to voluntarily contribute matchings to various data sources unless they receive some benefit from doing it.

We propose to do 3 things:

1. Automatically find candidate matchings between the columns of HTML tables to Freebase types or properties. This will provide a good set of matchings to display to the user and enable (2) and (3)
2. Allow the user to display related information from Freebase which is not present in the original HTML table (this will give users some incentive to use the system, and do the matching)
3. Allow the user to contribute the data contained in the table to Freebase with minimal effort

## Freebase

Freebase(Bollacker et al. 2008) is a large collaborative database of general human knowledge. It is publicly readable and writable using an HTTP-based query language.

[1] For a list of applications based on Freebase, see `http://www.freebase.com/view/freebase/metaweb_application`

Freebase represents data using objects and properties. Every object has properties `/type/object/name` and `/type/object/type`, so for any given surface form (string) we can easily query Freebase for all objects with that name, and generate a list of possible types. Since an object's *type* determines what other properties it possesses, we can then use these candidate types to get candidate properties to map to the other columns of the table.

For example, *Seattle* has the following types and associated properties (among many others):

- `/location/location`
  - `area = 369.073305723`
  - `containedby = [``King County'', ``Washington'', ``United States'']`
- `/location/statistical_region`
  - `population = 582454`

## Schema Matching

There has been much work on automatic schema matching (Rahm and Bernstein 2001). One example which shares some similarity is (Doan, Domingos, and Halevy 2001), in which an automatic procedure for matching source schemas to a shared *mediated schema* is learned from a few matching examples. In our work the mediated schema could be viewed as the Freebase ontology. Our work is different, however in that Freebase is already populated with a large amount of open-domain data. We therefore are able to rely on overlap in content between HTML tables and Freebase. Additionally, because every data element in Freebase contains associated type information, we can easily generate a list of candidate Freebase types for each table column, and generate a list of candidate properties for each column in relation to a type selected by the user.

(Tuchinda, Szekely, and Knoblock 2008) allow users to scrape data from Webpages interactively by exploiting the structure of the web page's DOM tree. In this respect they are not limited to extracting information only from tables, but instead can extract data from any website which contains some structure which can be exploited. These same DOM extraction techniques could be easily incorporated into our work as well. For the integration step, they make a similar assumption that the webpage will contain data which overlaps with data from another (already scraped) webpage. In

their case, however, they do not start with an existing populated database (such as Freebase), but instead have to start from scratch. From our experience, it is not very difficult to find tables which contain some overlap in information with Freebase, as Freebase contains a huge number of concepts on a wide variety of topics.

## Matching Procedure

The following procedure is used to map Freebase type schemas to HTML tables. More detail is provided in the following sections.

1. First of all we need to decide which Freebase schemas to consider matching to the table, and which columns represent keys. For each column containing string data, we take the entries to be entity names and query Freebase for possible types. For example, "Amazon" could have the types `/business/company` and `/geography/river`. For each candidate type a score is computed for how well it covers the values in the column. Note that although a given string (e.g. "Amazon") may be ambiguous, the set of all strings in a column of a *relational* table shouldn't have more than a few types in common. Additionally a name and type are usually sufficient to disambiguate a specific item.[2]
2. For the top $k$ (e.g. 10) types for each of these *(possible) key columns*, Freebase is queried to get all properties of each type, and the values of all these properties for the entities we've identified.
3. For each candidate type a *match score* is computed for each of its properties wrt each non-key column in the table using the following criteria (more details below):
   - String similarity between property name and column header
   - Average string similarity between property values and column elements (for string data)
   - Average percentage numeric difference between property values and column elements (for numeric data)
   - Cosine distance between the vectors of property values and column elements (for numeric data)
   - Score from #1 above (how well the type matches the key column)

### Scoring Column Types

To score how well type $t_i$ corresponds to column $j$ in the table, we use the following formula:

$$\text{TypeScore}_{ij} = \frac{T_j(t_i)}{\sqrt{\min(F(t_i), M)}}$$

Where $T_j(t_i)$ is the number of strings with type $t_i$ in column $j$ of the table, $F(t_i)$ is the number of entities with type $t_i$ in Freebase, and $M$ is a parameter which we (somewhat arbitrarily) set to 1000 to ensure the denominator doesn't grow too large.

[2]Additional mapped columns of the table could be used for further disambiguation if necessary, although we have not implemented this feature yet.

Note that we penalize types that occur very frequently in Freebase, as they are likely to match strings in the table by chance. For example, there are about 4.4 million entities of type `/music/track` in Freebase. We found that without this type of discounting, columns would very frequently be mistaken as containing music tracks.

### Match Criteria

In order to provide robust results even with missing table headers, or noisy data, we use a hybrid matching approach(Rahm and Bernstein 2001), combining both schema- and instance-level match criteria.

**Header/Property String Similarity** One indication of a good column/property matching is if the Freebase property name is similar to the HTML column header. This useful in finding a good match especially when there is no instance data available for a given attribute in Freebase. We use the Levenshtein Distance between the column header of the HTML table and the name of the Freebase property weighted by string length. As an example, in a table of buildings (see Figure 1), the "Structural Height" property of type `/architecture/structure` has large similarity to column header "height" in figure 1.

**Instance-Level String Similarity** If the instances in the column consist of string data (not numeric data or dates), then for each row of the HTML table, we can perform string comparisons between attributes of Freebase objects corresponding to that row, and the data in the table cell. For example, the *outflow* column in figure 2 contains the value "Mediterranean Sea" which is very similar to the value of the "Mouth" property of the `/geography/river` type for the instance "Nile" which corresponds to that row.

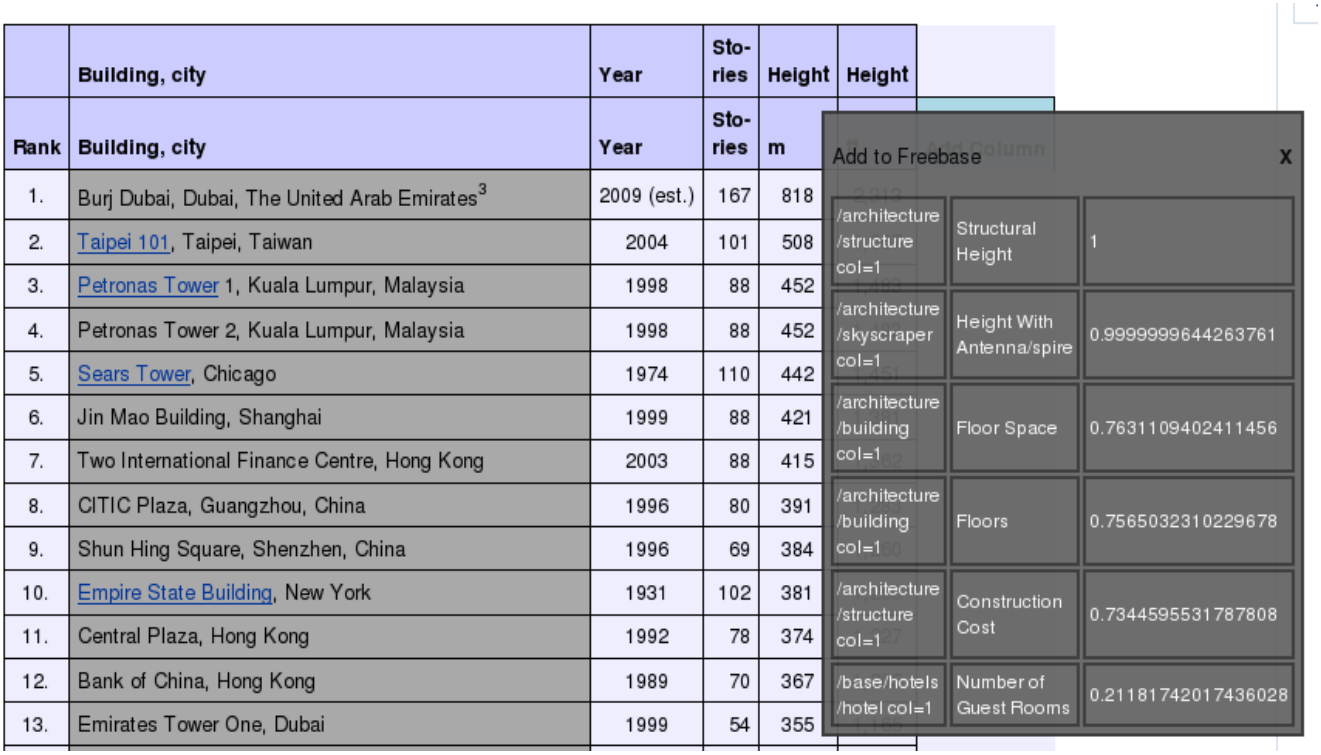
**Instance-Level Magnitude Similarity** For columns containing numeric data, a Freebase property is a good match if its values are roughly equal. Many types of numeric data can be close, but not exactly the same. E.g. populations of cities change from year to year. To determine how well attribute $a_i$ corresponds to column $j$ in the table, we use the following formula:

$$\text{MagScore}_{ij} = \frac{\sum_{k=1}^{n} \frac{|a_{ik} - c_{jk}|}{\min(|a_{ik}|, |c_{jk}|)}}{n}$$

This is simply the average, over all rows for which Freebase contains a value for the property $a_{ik}$, of the difference between $a_{ik}$ and $c_{jk}$ normalized by the minimum of the two. This is a *relative error* estimate, which ensures that differences between large values do not dominate the score.

**Instance-Level Cosine Distance** In some cases numeric data in a table may have strong correlation with data in Freebase, but have very different magnitude. For example, if Freebase contains height in meters, and the table contains height in feet, $\text{MagScore}_{ij}$ will be confused. If we can recognize that such a column maps to the correct attribute in Freebase, then it should be straightforward to convert from feet to meters, as we have several examples.

In order to measure the relative similarity between nu-



| Rank | Building, city | Year | Sto-ries | Height m | Height |
|---|---|---|---|---|---|
| 1. | Burj Dubai, Dubai, The United Arab Emirates[3] | 2009 (est.) | 167 | 818 | |
| 2. | Taipei 101, Taipei, Taiwan | 2004 | 101 | 508 | |
| 3. | Petronas Tower 1, Kuala Lumpur, Malaysia | 1998 | 88 | 452 | |
| 4. | Petronas Tower 2, Kuala Lumpur, Malaysia | 1998 | 88 | 452 | |
| 5. | Sears Tower, Chicago | 1974 | 110 | 442 | |
| 6. | Jin Mao Building, Shanghai | 1999 | 88 | 421 | |
| 7. | Two International Finance Centre, Hong Kong | 2003 | 88 | 415 | |
| 8. | CITIC Plaza, Guangzhou, China | 1996 | 80 | 391 | |
| 9. | Shun Hing Square, Shenzhen, China | 1996 | 69 | 384 | |
| 10. | Empire State Building, New York | 1931 | 102 | 381 | |
| 11. | Central Plaza, Hong Kong | 1992 | 78 | 374 | |
| 12. | Bank of China, Hong Kong | 1989 | 70 | 367 | |
| 13. | Emirates Tower One, Dubai | 1999 | 54 | 355 | |



Figure 1: We automatically recognize that the "Structural Height" property of the `/architecture/structure` type from Freebase maps to the *Height* column of the table

meric attributes and columns we use cosine distance, a standard distance measure between two vectors $a$ and $c$:

$$\cos(a, c) = \frac{a \cdot c}{\|a\|\|c\|}$$

## Combining Match Criteria

We combine our five match criteria $m_0, \ldots m_4$ using logistic regression:

$$\text{MatchScore} = \frac{1}{1 + e^{-\sum_{i=0}^{k} w_i m_i}}$$

Ideally we would learn the weights, $w_0, \ldots, w_4$, from positive and negative examples of matchings; so far we make the simplifying assumption that all matching criteria are equally important, assigning $w_i = 1 \forall i$. We have found this to work reasonably well in practice.

Note that by providing matchings users are not only directly contributing data to Freebase, but are also providing positive and negative examples of matchings which could be used labeled training data. This data could be used to learn weights for the match criteria using standard optimization techniques. So by providing matchings users could both immediately contribute data to Freebase in addition to contributing labeled training data which improves improves future match suggestions, and may even enable fully automatic matching on some high-confidence tables. Learning weights for the matching criteria is an item for future work.

## Contributing Data to Freebase

The output of the matching step is a score for each candidate property/column match. From this we can easily produce a ranked list of matches for each column in the table. The user can quickly view these, and select the correct property or determine that there is no good match in Freebase[3]

## Augmenting Tables with Data from Freebase

Once we have a good list of candidate types for the key columns, we can offer the user the option to display additional columns not originally present in the HTML table. This provides direct benefit to users; we feel that it will give users incentive to use the system and contribute data to Freebase.

For example, consider the New York times article in figure 3. A list of "The Best 1,000" movies is presented with dates. What if a user wants to select a movie to watch, and would like to know what language each movie is in, who directed it, or other similar information in order to make the decision?

Because our system can automatically detect that column 1 of this table corresponds to the Freebase type

[3]It would be easy to add an option for the user to add a new property to one of the types found for the `id` column. We chose not to, as this is more a schema design problem than a data entry one.

Also, user's could add new types to existing objects, this too is an item for future work.

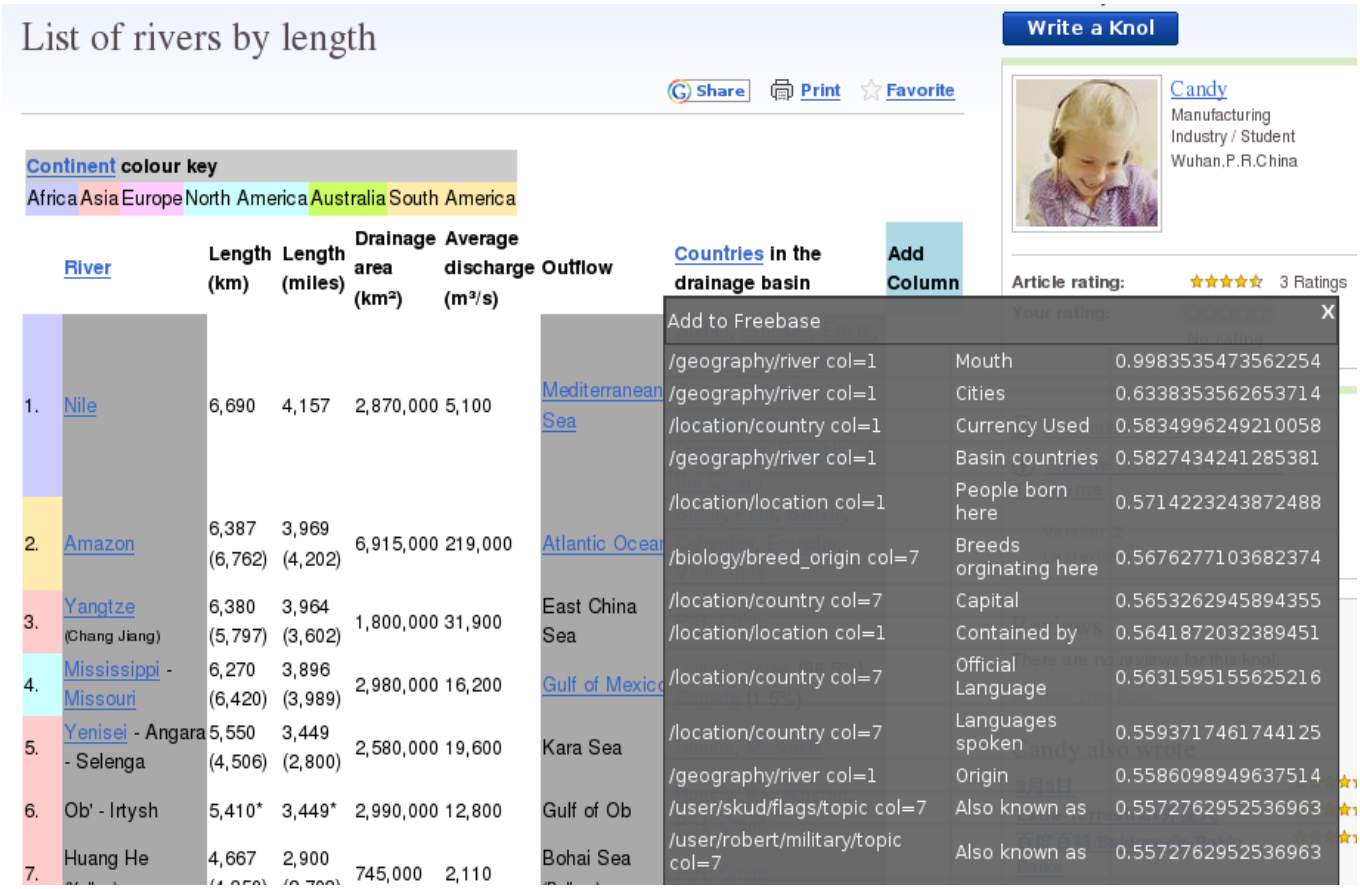



Figure 2: The "Mouth" attribute of `/geography/river` maps to the "Outflow" column in the table

`/film/film`, the user can choose to display any properties for those films present in Freebase. This provides direct benefit to the user, as they can choose what attributes to display. They are not limited to seeing only the static information chosen by the authors of the page.

## Implementation

We implemented our entire system in Javascript using the Greasemonkey[4] firefox extension (it should be trivial to convert our system to a standalone firefox extension). To access Freebase we query the public *read* and *write* APIs using the Metaweb Query Language (MQL)[5].

Implementing our entire system as a browser plugin which directly queries Freebase has the advantage that it is easy to distribute, and does not require a server to process the data. It is worth noting, that this implementation is somewhat slow on large HTML tables due to the time needed to query Freebase in addition to CPU time spent computing cosine and string edit distances. The tables in our evaluation took about 15 seconds on average to load. We feel confident, however that this time could be substantially reduced with sufficient engineering effort.

## Evaluation

In order to evaluate our system, we measured performance at matching table columns to attributes in Freebase. Note that this also indirectly measures performance at identifying types for columns, as this is part of the matching criteria.

We collected a set of 12 tables which met the following criteria:

1. They contained relational data (each row represents a tuple, each column an attribute).
   - This requirement eliminates tables which are used for formatting purposes, or tables where rows and columns have no significance.
2. They have some overlap with data contained in Freebase.
   - We feel it should be fairly obvious to users from the lack of related types and attributes if a table does not correspond to any data in Freebase.

These 12 tables contained a total of 61 non-key columns, each of which we hand-labeled with a "ground-truth" Freebase property. Out of the 61 columns, we found 39 (64%) corresponded to an attribute existing in Freebase. Many of the other columns represented attributes missing from Freebase which could be added; again this is a schema design issue which our system does not currently handle[6].

Out of the 39 columns for which a good match exists, our system found the correct property as its first guess for 30 of them (or about 77% of the time). The rest of the matches were mostly in the top 10 results presented to the user. A histogram of the rank of the correct match is displayed in figure 4.

[4] `http://www.greasespot.net/`

[5] http://mql.freebaseapps.com/

[6] Users can easily use the Freebase schema editor to add missing properties, at which point our system could populate the instances.

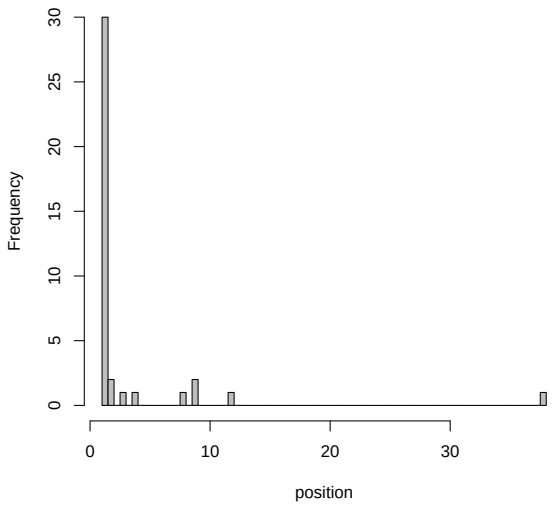



Figure 4: Histogram of the rank of correct match for each of the 39 columns for which such a match exists. For 77% of the columns the correct match was the first guess.

## Conclusions

We have presented a system for semi-automatically mapping between HTML tables and Freebase, an online database. Because data integration is rarely 100% accurate, some human supervision is needed in the process. We hope our system will allow ordinary surfers to contribute large volumes of structured data easily by quickly verifying our mapping results. In addition, we provide some incentive for using our system, by allowing users to view additional data not present in the original table, but available in Freebase.

## References

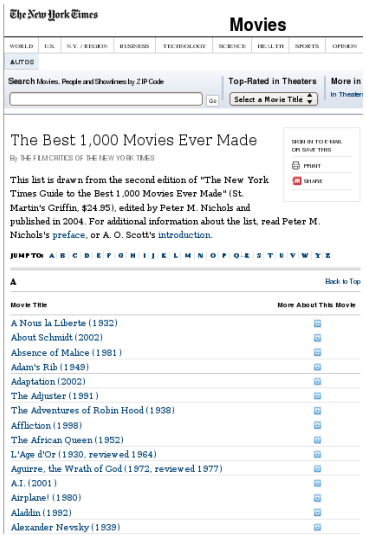

Auer, S.; Bizer, C.; Kobilarov, G.; Lehmann, J.; and Ives, Z. 2007. Dbpedia: A nucleus for a web of open data. In *In 6th Intl Semantic Web Conference, Busan, Korea*, 11–15. Springer.

Bollacker, K.; Evans, C.; Paritosh, P.; Sturge, T.; and Taylor, J. 2008. Freebase: a collaboratively created graph database for structuring human knowledge. In *SIGMOD '08: Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, 1247–1250. New York, NY, USA: ACM.

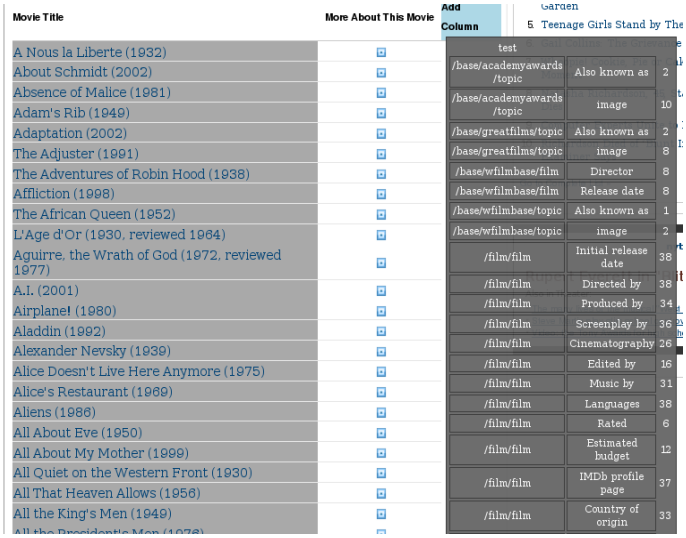
Cafarella, M. J.; Halevy, A. Y.; Wang, D. Z.; 0002, E. W.; and Zhang, Y. 2008. Webtables: exploring the power of tables on the web. *PVLDB* 1(1):538–549.

Doan, A.; Domingos, P.; and Halevy, A. 2001. Reconciling schemas of disparate data sources: A machine-learning approach. In *In SIGMOD Conference*, 509–520.

Hoffman, R.; Amershi, S.; Fogarty, J.; Patel, K.; Weld, D.; and Wu, F. 2009. "amplifying community content creation with mixed-initiative information extraction". In




(a)




(b)


(c)

Figure 3: (a) A table of movies (b) Displaying a list of Freebase properties that can be shown (c) Table has been augmented with directors and writers from Freebase

*ACM Conference on Human Factors in Computing Systems (CHI 2009)*.

Rahm, E., and Bernstein, P. A. 2001. A survey of approaches to automatic schema matching. *The VLDB Journal* 10(4):334–350.

Tuchinda, R.; Szekely, P.; and Knoblock, C. A. 2008. Building mashups by example. In *IUI '08: Proceedings of the 13th international conference on Intelligent user interfaces*, 139–148. New York, NY, USA: ACM.

Wu, F., and Weld, D. S. 2007. Autonomously semantifying wikipedia. In *CIKM '07: Proceedings of the sixteenth ACM conference on Conference on information and knowledge management*, 41–50. New York, NY, USA: ACM.