DataXFormer: An Interactive Data Transformation Tool

John Morcos[1] Ziawasch Abedjan[2] Ihab F. Ilyas[1]
Mourad Ouzzani[3] Paolo Papotti[3] Michael Stonebraker[2]
[1] University of Waterloo [2] MIT CSAIL [3] Qatar Computing Research Institute
{jmorcos,ilyas}@uwaterloo.ca {abedjan,stonebraker}@csail.mit.edu {mouzzani,ppapotti}@qf.org.qa

ABSTRACT

While syntactic transformations require the application of a formula on the input values, such as unit conversion or date format conversions, semantic transformations, such as *zip code to city*, require a look-up in some reference data. We recently presented `DataXFormer`, a system that leverages Web tables, Web forms, and expert sourcing to cover a wide range of transformations. In this demonstration, we present the user-interaction with `DataXFormer` and show scenarios on how it can be used to transform data and explore the effectiveness and efficiency of several approaches for transformation discovery, leveraging about 112 million tables and online sources.

Categories and Subject Descriptors

H.2.5 [**Database Management**]: Heterogeneous Databases

Keywords

data integration, data enrichment, data transformation, web tables, web forms, wrapper, deep web

1. INTRODUCTION

Several data analytics tasks require integrating various heterogeneous data sources, where the same or highly related information might be expressed in different forms. While there has been a considerable amount of research on schema matching and schema mapping, the related task of transforming the actual values from one representation to the target representation has been neglected. Examples of data transformation tasks include mapping stock symbols to company names, changing date format from `MM-DD to DD-MM` and replacing cities by their countries. While some transformations such as *liters to gallons* can be performed by applying a formula or a program on the input values, others, such as *company name to stock symbol* and *event to date*, require finding the mappings between the input and output values in a repository of reference data. We refer to the former type of transformations as "syntactic transformations" and to the latter as "semantic transformations".

Syntactic transformations are supported by many tools, including the popular MS-Excel, Google Spreadsheets and the more recent Wrangler [7]. However, to the best of our knowledge, no automatic system or tool is available that significantly covers the class of semantic transformations, Semantic transformations are prevalent in many real-world integration tasks, as witnessed in workloads of the data curation tool Tamr [9] and other data vendor companies.

When companies need to explore or aggregate their data, they create dynamic views that join multiple databases. Here, it is desirable to dynamically discover the desired transformations that allows them to join multiple sources on a unified attribute. Furthermore, useful mappings, such as *US Dollars to EUR*, *genome to coordinates*, *location to temperature*, change over time and requires users to look-up in a continuously updated repository, making previously acquired reference datasets obsolete.

The main reason of why semantic transformations have been neglected so far is that they cannot be computed by solely considering the input values and applying a formula or a string operation on them. Rather, the required transformations can often be found in a mapping table that is either explicitly available to the application (e.g., as a dimension table in a data warehouse) or hidden behind a transformation service or a Web form.

While collecting the adequate reference data for one specific transformation task is doable, it requires tedious crawling and curation exercises. This process cannot easily scale for a large number of transformation tasks covering the long tail of domains and topics. Usually, the subject matter expert in a company who is interested in converting its data does not have the skills to obtain the reference data or to code the transformation formula. Considering that the Web provides a large repository of resources, it is useful to have a service that can leverage these resources for on-demand transformation tasks.

At CIDR 2015, we presented `DataXFormer` [1], a system that leverages Web tables and Web forms to perform syntactic and semantic transformations. `DataXFormer` expects a transformation task specified as a variable-sized collection of examples, giving the entire input X and some examples of the desired output Y. The following are examples for the *airport code to city name* transformation: $\{(LHR, London), (ORD, Chicago), (CDG, Paris)\}$. In most cases, the user provides the column labels of the

SIGMOD'15, May 31–June 4, 2015, Melbourne, Victoria, Australia.
Copyright © 2015 ACM 978-1-4503-2758-9/15/05 ...$15.00.
http://dx.doi.org/10.1145/2723372.2735366.

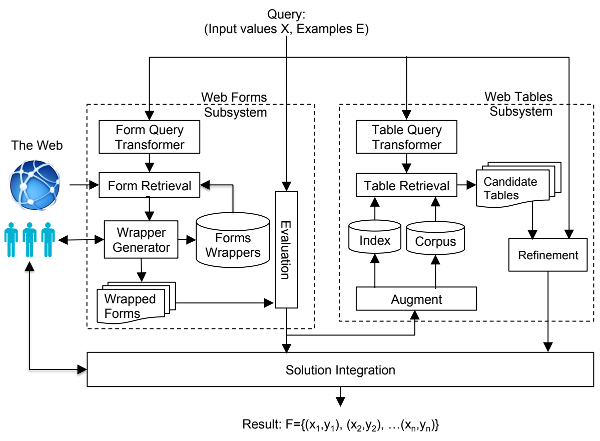


Figure 1: `DataXFormer` architecture

input and output values for the desired transformation, which we denote by I_X and I_Y, e.g., $I_X = airportcode$ and $I_Y = city$.

In this demonstration, we show the various features of `DataXFormer` and how it finds transformation answers in Web tables and Web forms. We also show how `DataXFormer` interacts with the user to enrich its corpora, provide better answers and unlock previously unusable Web forms. In particular, we trace the intermediate steps of `DataXFormer` to find and wrap Web forms in an interactive manner. An initial version of `DataXFormer` is already available at `http://dataxformer.org`.

2. DataXFormer OVERVIEW

`DataXFormer` (see Figure 1) consists of two complementary transformation engines: one based on locally-stored static Web tables and another based on dynamically discovered Web forms. The user starts by submitting a query, such as the query that transforms airport codes to city names. `DataXFormer` converts the user query into the corresponding internal forms for each of the retrieval components, which in turn return candidate Web tables and Web forms.

`DataXFormer` dispatches the transformation query simultaneously to both transformation engines. While the two engines differ in their retrieval interface, response time and coverage, we are currently taking the straightforward approach of engaging both systems simultaneously. In a final step, the *solution integration* component presents the best effort results from both sources for a given query. Here, the user can intervene to choose among different transformation discovery approaches and results. We present and rank the results of each subsystem separately because of the significant difference in response time. Note that the Web forms subsystem has a higher response time than the local Web tables subsystem because it has to submit a request to a remote server to transform each value. Having an automatic mechanism for merging the results of both sources is a future effort, considering that they are good at different transformation tasks.

2.1 Web tables subsystem

The task of the tables subsystem is to identify all the tables that contain a minimum number of the user examples in one of their column projections. We refer to this minimum threshold as τ. Our experiments showed that $\tau = 2$ is a reasonable threshold for filtering irrelevant tables while maintaining very high recall [1].

A major challenge is to maintain the interactivity of the system through fast response times while being able to sift through 112 million tables in the corpus to find matches for the given examples. Thus, we maintain an inverted index that points to the columns and tables that contain the X or Y values. Since Web tables are heterogeneous and differ in schema and size and some even lack column labels, we store the tables within a universal main table (relation *Cells* as in Figure 2); every cell of a Web table is represented by a tuple that records the table, column and row IDs, along with the tokenized, stemmed form of its value (*tokenized*). Using this schema, we can simulate an inverted index by having an index on the *tokenized* column. The relation is ordered by *tableid*, *columnid*, *rowid*, simultaneously achieving two advantages: (i) every column from a web table is stored contiguously, and (ii) the space requirement of this schema is alleviated by compression, which is provided by most modern column-stores.

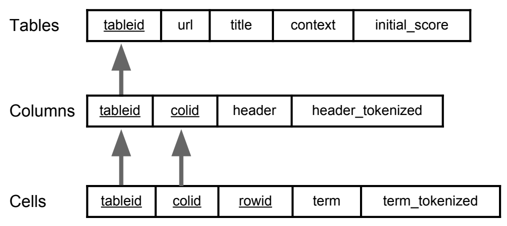


Figure 2: Schema for storing Web tables in a column-store

In the current prototype, we store the Web tables in a multi-node Vertica instance. Vertica employs projections on tables in place of an inverted index. A projection is a specialized materialized view that is efficient to maintain and load. We use a projection on *Cells* that is sorted on the tokenized values. Additionally, `DataXFormer` includes a similarity index to retrieve all existing similar representations of given X and Y values within a given edit distance threshold. Using a projection, the *table retrieval* component retrieves a set of relevant candidate tables for the given user query and verifies the coverage of the candidate tables with respect to the query examples. If the coverage is above the user-defined threshold, `DataXFormer` extracts the rest of the required transformation values.

We now proceed to describe the transformation discovery approaches that use `DataXFormer`'s Web table repository.

Direct matching The fastest and simplest way to find the required transformation in the tables is to find tables that contain the required minimum number of given examples, as evidence for the two columns that encode the transformation. While the look-up operation is very fast, it is very sensitive to the choice of the initial examples. In Figure 3, we refer to the direct matching as *iteration 1* of the look-up operation, which matches Tables 1 and 2, with transformations for the airport codes *FRA*, *LAX*, and *BOS*.

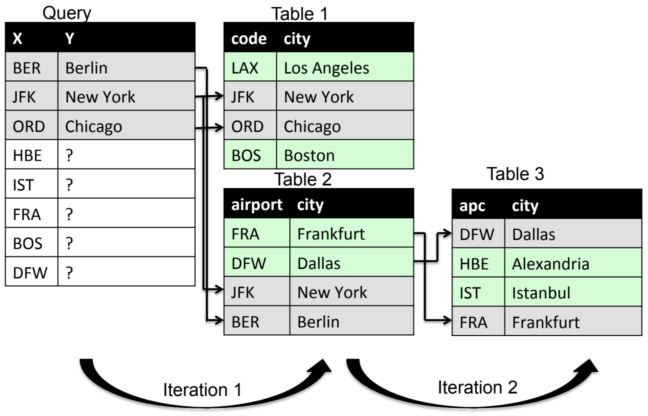


Figure 3: Iterations for transformation discovery for $\tau = 2$

Iterative matching The direct matching approach is fast and simple. However, the transformation entries may sometimes be fragmented over several tables (Figure 3). Most Web tables were designed for human consumption, and thus, large tables are usually fragmented over separate tables on different Web pages in a given Web site. Since the input examples are very few, only a small fraction of all the tables that encode the transformation might match the transformation query. In Figure 3, our query examples only match tables that focus on airports in the USA, and cannot find the transformation for *IST* and *HBE* in those tables.

To increase the recall, the iterative matching looks for indirect matches by using newly found results as examples for the next iteration. As illustrated in Figure 3, the new iteration uses the transformation results for *FRA* and *DFW* in Table 2 as new examples, which enables `DataXFormer` to reach Table 3 and find the transformations for *HBE* and *IST*. Iterative matching increases the recall at the cost of a longer execution time. There might also be a reduction in precision as non-evaluated transformation results are used as examples. We solve this issue in the reconciliation phase where different results are ranked based on their redundancy and the authoritativeness of their sources.

Fuzzy matching To further increase recall, the user might decide to relax the matching of values, accounting for typos and slight changes in representation. Our basic approach already tolerates some fuzziness by considering tokenization and stemming of transformation values. However, variations such as typos or abbreviations of a term cannot be captured by stemming and tokenization. For example, "Washington DC" and "Washington D.C." can only be matched by tolerating a certain degree of edit distance, since most tokenization techniques would assume that "DC" is a token by itself, while "D.C." consists of two tokens. We apply the popular Levenshtein similarity measure to capture some of these fuzzy matches. Fuzzy matching takes a step further towards increasing recall, at the expense of the execution time.

Reconciling different transformation results Our transformation results may stem from multiple different tables. Hence, for some X values, multiple Y may be retrieved. To reconcile these contradicting results, `DataXFormer` applies an expectation-maximization approach [5] that assigns scores to transformation results, which in turn affect the perceived authoritativeness of the tables providing them. Initially, the model takes into account the prevalence of the number of correct examples in a table. Furthermore, the model takes into account how authoritative a table is, based on its initial score. The initial score of a table changes based on user-feedback. Over time, we capture whether users accept or reject transformation results and use their evaluation to enhance or decrease initial scores of the corresponding tables. Also it is possible to manually add authoritative tables to the repository. More details on the reconciliation process of `DataXFormer` and how it converges can be found in our previous paper [1].

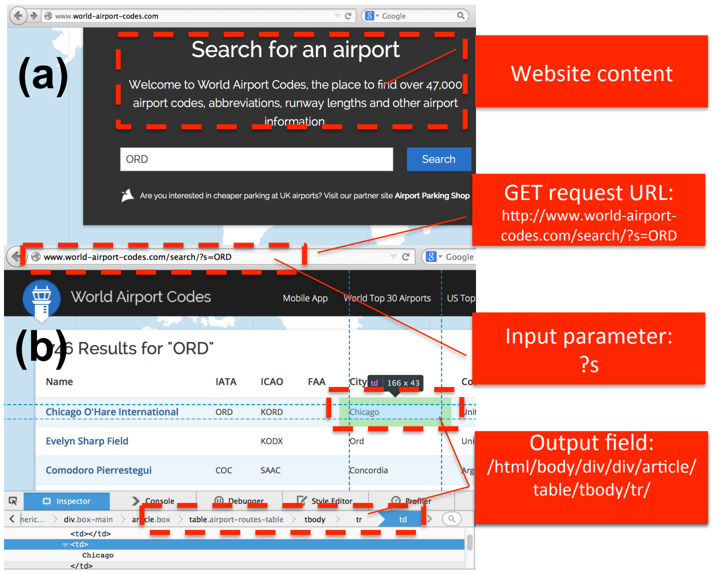
2.2 Web forms subsystem

A Web form allows user-interaction through different fields and buttons. The input values given by a user are either processed directly on the client side through `JavaScript` or are sent as an HTTP request (GET or POST) to a back-end server for further processing. Figure 4a illustrates a Web form for discovering location attributes of a given airport code. Putting *ORD* into the search field on Web page (a) retrieves the result Web page (b) that contains the corresponding city name *Chicago*.

As with Web tables, we assume a Web form is relevant if it covers at least τ of the example transformations. There are two main challenges in using Web forms: (1) as there is no common repository of Web forms, we have to look for them through crawling billions of Web pages; and (2) a new Web form appears as a black box. This means that an invocation strategy (i.e., wrapper) has to be developed to use the form for producing the desired transformations. It has been shown [4] that both tasks are very hard, even with human intervention. `DataXFormer` tackles both challenges using search engines and a Web browser simulator that allows to analyze the HTML code of a Web form. Additionally, the user is involved whenever any of the tasks fail.

Web form retrieval `DataXFormer` maintains a local repository of previously used Web forms. The repository is organized as a document store where each document represents one wrapper. Using the document store is suitable for this task because we also store and index the textual content of Websites that contain the specific Web form. If `DataXFormer` fails to find a relevant form in the repository, it uses a search engine to find forms online. In our preliminary experimental results [1], we noticed that the keyword query "I_X `to` I_Y" has a high success rate for finding a page containing a transformation form in its top results. By examining the search engine results, `DataXFormer` identifies candidate forms. Then, for each candidate form, we generate a "wrapper" to be able to query the form and to obtain the transformation values. In case `DataXFormer` fails to generate a wrapper or to find a relevant Web form, the user is asked for help. The wrapped forms are invoked using the input examples and are evaluated based on their coverage in the *evaluation* step. Candidate Web forms are then queried using the examples from the user query.

Wrapping Web forms To wrap Web forms, we have to simulate a Web browser and probe the forms using the given examples to identify the relevant input field to fill in the X values, the output field that contains the desired transformation result Y, and the request method for invoking the


(a)
(b)
Website content
GET request URL: http://www.world-airport-codes.com/search/?s=ORD
Input parameter: ?s
Output field: /html/body/div/div/article/table/tbody/tr/
www.world-airport-codes.com
www.world-airport-codes.com/search/?s=ORD
Search
ORD
World Airport Codes
Mobile App
World Top 30 Airports
Inspector
Console
Debugger
Style Editor
Profiler

Search for an airport

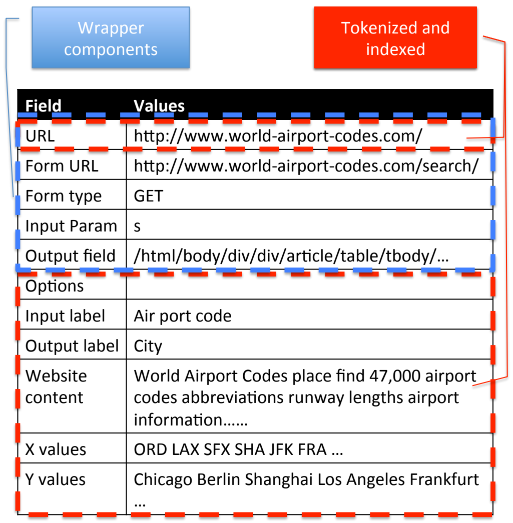
Welcome to World Airport Codes, the place to find over 47,000 airport codes, abbreviations, runway lengths and other airport information

Are you interested in cheaper parking at UK airports? Visit our partner site Airport Parking Shop

46 Results for "ORD"

Name	IATA	ICAO	FAA	City
Chicago O'Hare International	ORD	KORD		Chicago
Evelyn Sharp Field		KODX		Ord
Comodoro Pierrestegui	COC	SAAC		Concordia

(a) Web form components that need to be stored


Wrapper components
Tokenized and indexed

Field	Values
URL	http://www.world-airport-codes.com/
Form URL	http://www.world-airport-codes.com/search/
Form type	GET
Input Param	s
Output field	/html/body/div/div/article/table/tbody/...
Options	
Input label	Air port code
Output label	City
Website content	World Airport Codes place find 47,000 airport codes abbreviations runway lengths airport information......
X values	ORD LAX SFX SHA JFK FRA ...
Y values	Chicago Berlin Shanghai Los Angeles Frankfurt ...

(b) Web form wrapper as a document in the repository

Figure 4: Web form wrapper components

form. Current wrapper generation approaches [4, 8] invoke the form using all possible combinations of input fields, select fields, and radio buttons, and use the results to identify the semantics of the various components of the form. In the current prototype, we apply some heuristics to reduce the number of combinations; for example, we only consider text and text area fields as input fields for our X values, and we choose radio button options that resemble our attribute labels or denote the transformation direction. In Figure 4a, we highlight the visible relevant components such as the Website content, the GET request URL, the input parameter s, and the XPath for the output value.

Once the XPath to our given example is discovered on the output result page, `DataXFormer` invokes the wrapped form using the remaining input examples and evaluates its coverage in the *evaluation* step. If the user is satisfied with the Web form results, she can trigger the *Augment* component to store the Web form results as a new table in the table repository. Finally, `DataXFormer` stores all components of a successfully wrapped Web form as illustrated in Figure 4b.

Some of the stored fields (framed red in Figure 4b) contain meta-data that is indexed to effectively retrieve the wrapper for a given transformation query. For example, *Website content* and example X values are tokenized and indexed. Other fields that are framed in blue in Figure 4b store the configuration parameters of the wrapper, such as the *Form URL* and *Input Param.*

2.3 Datasets and Implementation Setup

`DataXFormer` uses a Vertica column-store to store the Web tables and a Lucene index that indexes successfully wrapped Web forms. Initially, the Web tables corpus was filled with the Dresden Web Table Corpus [1], a collection of about 112 million data tables extracted from the Common Crawl [2]. We further cleaned the dataset by removing cells that contained comments or other long text content which does not serve our purpose. The disk usage of the Vertica storage comprises about 350 GB on 4 nodes.

The Web forms are stored as documents, where each document contains the URI of a Web form and all its relevant fields, to allow reconstructing the generated wrapper of the Web form. Furthermore, we store some meta-data, such as the textual content of the Website that provides the Web form, as well as a limited set of previously performed example transformations. Some of the wrappers refer to the same Web form with a different configuration, e.g., both directions for *km to/from miles.*

3. DEMONSTRATION

An initial version of `DataXFormer`, which is already at `http://dataxformer.org`, answers transformation queries through direct and iterative matching on Web tables and through indexed Web forms.

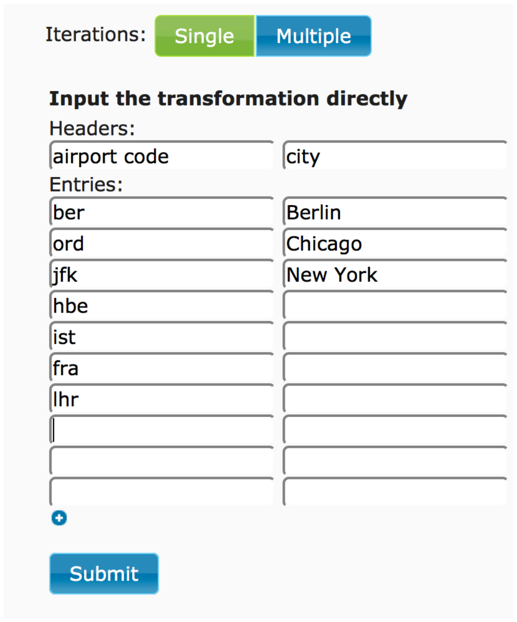
The demonstration's audience can come up with its own transformation queries for `DataXFormer`. For example a user might be interested in discovering transformations of her favourite *athlete to associated club*. Additionally, we will provide and describe some prepared transformation queries from our previous benchmark [1], such as *airport code to city*, *company ticker to company name*, and *EUR to USD*.

3.1 Using DataXFormer

A user can submit a query either by uploading a CSV file containing the two columns X and Y or by filling in the examples through a simple table form (see Figure 5).

Then, the user can start the transformation engine, which will initiate the selected transformation discovery algorithm (direct, indirect, or fuzzy matching) over the Web tables and the Web form retrieval. `DataXFormer` dispatches both subsystems in parallel. Figure 6 presents the retrieved indirect match results for the query. The selected values in the

[1]`https://wwwdb.inf.tu-dresden.de/misc/dwtc/`
[2]`http://commoncrawl.org/`


Iterations: Single Multiple

Input the transformation directly

Headers:

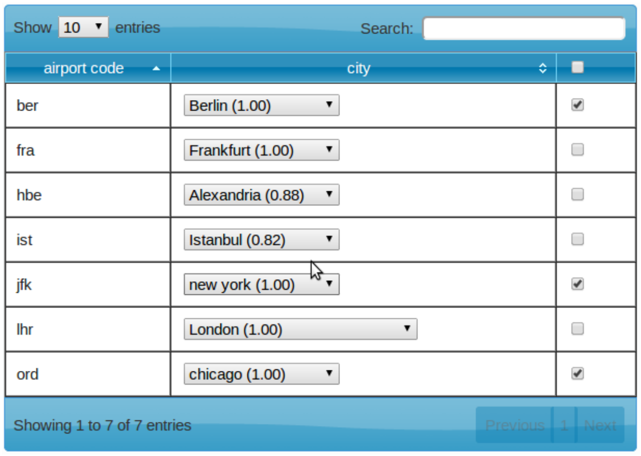
airport code	city

Entries:

ber	Berlin
ord	Chicago
jfk	New York
hbe	
ist	
fra	
lhr	

Submit

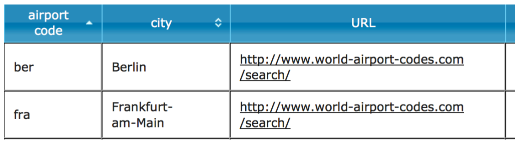
Figure 5: Our transformation query aims at transforming airport code to the city location. Here, we provided the corresponding cities for three of the airport codes.


Show 10 entries Search:

airport code	city	☐
ber	Berlin (1.00)	✓
fra	Frankfurt (1.00)	☐
hbe	Alexandria (0.88)	☐
ist	Istanbul (0.82)	☐
jfk	new york (1.00)	✓
lhr	London (1.00)	☐
ord	chicago (1.00)	✓

Showing 1 to 7 of 7 entries Previous 1 Next

Figure 6: Tranformation results of the iterative matching approach



airport code	city	URL
ber	Berlin	http://www.world-airport-codes.com /search/
fra	Frankfurt-am-Main	http://www.world-airport-codes.com /search/

Figure 7: `DataXFormer` found a Webform that can find the desired cities.

city column are the transformation results with the highest scores, which are displayed in brackets. At the same time, `DataXFormer` also presents the results that were produced by a Web form (Figure 7).

If the user is not satisfied with the results, she might use the drop down boxes to choose from alternative transformations, if available (see Figure 8). Note that the results are ranked according to their confidence score assigned by `DataXFormer`'s expectation-maximization approach [1]. Upon choosing an option, pop-up widgets provide further background information, such as the number of tables that agreed on this specific transformation. Furthermore, the user can display the actual content of a table to understand the provenance of a transformation result (see Figure 9).

Alternatively, the user can resend the transformation query with a subset of successfully performed transformations. Depending on the initial matching algorithm, the user can also choose to enhance the transformation results by changing the matching algorithm on the Web tables.

If the user is satisfied, she can save the complete result set or a subset of the results by checking the corresponding check boxes and add it to the corpus. A table that is stored by a user has a higher initial score. We will demonstrate the evolution of the corpus by resubmitting the initial query and showing that the results and scores have been changed.

3.2 Tracing Web form discovery and user-based wrapper induction

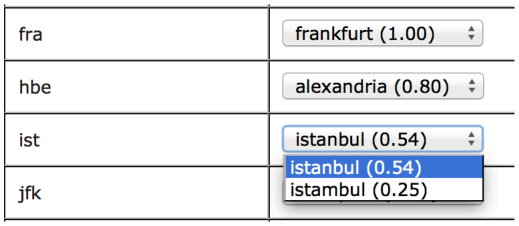
Our demonstration also includes a walkthrough of the Web form subsystem by showing the intermediate steps and results. We will show how `DataXFormer` leverages the search engine, and which aspects of a Website matter for the automatic wrapper induction. Furthermore, we show a scenario where the user can intervene in the wrapper induction without having any background in Web programming. `DataXFormer` analyzes a given website, extracts its HTML elements, and simulates its Web forms within `DataXFormer`'s own framework. In Figure 10 `DataXFormer` identifies two Web forms on `http://world-airport-codes.com`, and renders an interactive wrapper induction interface. Note, that `DataXFormer` identifies human-readable descriptions. Popups will show the corresponding original HTML code. The user can then continue to configure the selected form, by choosing the correct input field and form options. `DataXFormer` captures the user's interaction with the simulated form and uses the selected parameters to wrap the original Web form for the transformation task.

4. RELATED WORK

The closest tool to `DataXFormer` is Google Spreadsheets [3]. The autofill feature uses machine learning to predict the transformation values of a column given a few examples. In contrast, `DataXFormer` does not try to explicitly define the transformation, but rather finds the necessary look-up tables in a corpus of Web tables and Web forms. Since Google Spreasheets tries to interpolate the transformation solely from the input, it cannot provide semantic transformations, such as *"airport code to city"*, while `DataXFormer` can since it uses external knowledge.

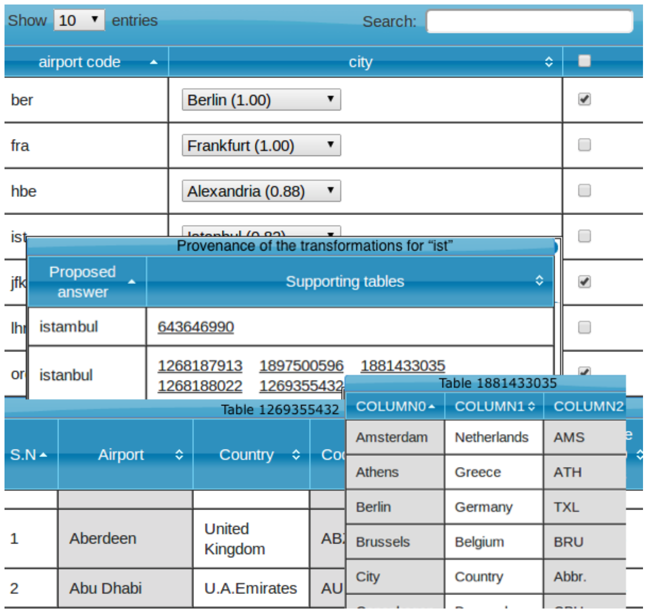
InfoGather [10] uses Web tables for attribute discovery and entity augmentation. While the two approaches are quite similar, InfoGather relies on pre-computing the full semantic-match graph for the whole corpus, while `DataX-`

[3]`http://spreadsheets.google.com`



fra	frankfurt (1.00)
hbe	alexandria (0.80)
ist	istanbul (0.54)
jfk	istanbul (0.54) istambul (0.25)

Figure 8: The user can choose between different transformation results, which are ranked according to their confidence score



Show 10 entries Search:

airport code	city	
ber	Berlin (1.00)	✓
fra	Frankfurt (1.00)	
hbe	Alexandria (0.88)	
ist	Istanbul (0.82)	
jfk		✓
lhr		
or		✓

Provenance of the transformations for "ist"

Proposed answer	Supporting tables
istambul	643646990
istanbul	1268187913 1897500596 1881433035 1268188022 1269355432

Table 1269355432

S.N	Airport	Country	Co
1	Aberdeen	United Kingdom	AB
2	Abu Dhabi	U.A.Emirates	AU

Table 1881433035

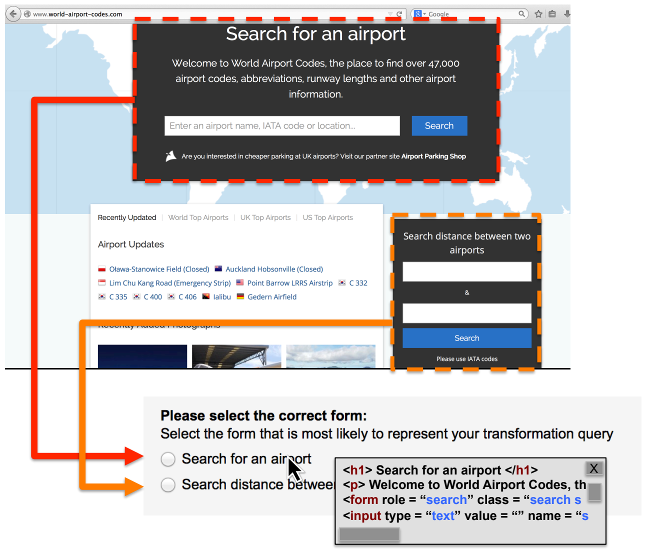
COLUMN0	COLUMN1	COLUMN2
Amsterdam	Netherlands	AMS
Athens	Greece	ATH
Berlin	Germany	TXL
Brussels	Belgium	BRU
City	Country	Abbr.

Figure 9: DataXFormer shows the provenance of a transformation result. Note that some tables do not have column headers.

Former finds the required tables on the fly, and leverages Web forms as well.

Wrangler [7] provides an easy interface to specify scripts of data transformation. It supports the concept of semantic roles as well, providing additional functions to be included in the scripts. DataXFormer requires minimal user interaction, as it locates Web tables and forms automatically. Similarly to Google Spreadsheets, Wrangler cannot provide transformations that rely on external knowledge.

Many proposed systems focus on answering keyword queries over relational tables by efficiently generating candidate networks of joined tuples to form answers, with search terms from the query spanning multiple records across multiple tables [2, 3, 6]. DataXFormer differs from these systems in that it tries to find tables that provide a transformation given a few examples for the transformation, rather than building candidate tuple networks.



www.world-airport-codes.com
Google
Search for an airport
Welcome to World Airport Codes, the place to find over 47,000 airport codes, abbreviations, runway lengths and other airport information.
Enter an airport name, IATA code or location...
Search
Are you interested in cheaper parking at UK airports? Visit our partner site Airport Parking Shop
Recently Updated | World Top Airports | UK Top Airports | US Top Airports
Airport Updates
Oława-Stanowice Field (Closed) Auckland Hobsonville (Closed)
Lim Chu Kang Road (Emergency Strip) Point Barrow LRRS Airstrip C 332
C 335 C 400 C 406 Ialibu Gedern Airfield
Search distance between two airports
&
Search
Please use IATA codes

Please select the correct form:
Select the form that is most likely to represent your transformation query

- ◯ Search for an airport
- ◯ Search distance betwee

```
<h1> Search for an airport </h1>
<p> Welcome to World Airport Codes, th
<form role = "search" class = "search s
<input type = "text" value = "" name = "s
```

Figure 10: DataXFormer identifies the forms on a web page and presents them as radio options.

5. REFERENCES

[1] Z. Abedjan, J. Morcos, M. Gubanov, I. Ilyas, M. Stonebraker, P. Papotti, and M. Ouzzani. Dataxformer: Leveraging the web for semantic transformations. In *CIDR*, 2015.

[2] B. Aditya, G. Bhalotia, S. Chakrabarti, A. Hulgeri, C. Nakhe, P. Parag, and S. Sudarshan. Banks: Browsing and keyword searching in relational databases. In *VLDB*, pages 1083–1086, 2002.

[3] S. Agrawal, S. Chaudhuri, and G. Das. Dbxplorer: A system for keyword-based search over relational databases. In *ICDE*, pages 5–16, 2002.

[4] L. Barbosa and J. Freire. An adaptive crawler for locating hidden-web entry points. In *WWW*, pages 441–450, New York, NY, USA, 2007.

[5] A. P. Dawid and A. M. Skene. Maximum likelihood estimation of observer error-rates using the em algorithm. *Applied statistics*, pages 20–28, 1979.

[6] V. Hristidis and Y. Papakonstantinou. Discover: Keyword search in relational databases. In *VLDB*, pages 670–681, 2002.

[7] S. Kandel, A. Paepcke, J. Hellerstein, and J. Heer. Wrangler: Interactive visual specification of data transformation scripts. In *CHI*, pages 3363–3372, New York, NY, USA, 2011.

[8] J. Madhavan, D. Ko, L. Kot, V. Ganapathy, A. Rasmussen, and A. Halevy. Google's deep web crawl. *PVLDB*, 1(2):1241–1252, Aug. 2008.

[9] M. Stonebraker, D. Bruckner, I. F. Ilyas, G. Beskales, M. Cherniack, S. B. Zdonik, A. Pagan, and S. Xu. Data curation at scale: The data tamer system. In *CIDR*, 2013.

[10] M. Yakout, K. Ganjam, K. Chakrabarti, and S. Chaudhuri. Infogather: Entity augmentation and attribute discovery by holistic matching with web tables. In *SIGMOD*, pages 97–108, New York, NY, USA, 2012.