DataXFormer: Leveraging the Web for Semantic Transformations

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

ABSTRACT

Data transformation is a crucial step in data integration. While some transformations, such as *liters to gallons*, can be easily performed by applying a formula or a program on the input values, others, such as *zip code to city*, require sifting through a repository containing explicit value mappings. There are already powerful systems that provide formulae and algorithms for transformations. However, the automated identification of reference datasets to support value mapping remains largely unresolved. The Web is home to millions of tables with many containing explicit value mappings. This is in addition to value mappings hidden behind Web forms. In this paper, we present `DataXFormer`, a transformation engine that leverages Web tables and Web forms to perform transformation tasks. In particular, we describe an inductive, filter-refine approach for identifying explicit transformations in a corpus of Web tables and an approach to dynamically retrieve and wrap Web forms. Experiments show that the combination of both resource types covers more than 80% of transformation queries formulated by real-world users.

1. INTRODUCTION

Organizations need to integrate various heterogeneous data sources, where the same or highly related information might be expressed in different ways. To allow for tasks such as schema integration and record linkage, data transformation tools provide mappings across these pieces of information. Examples include mapping stock symbols to company names, changing date formats from `MM-DD to DD-MM`, or 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 several tools, including the popular MS Excel and the more recent Wrangler [17]. However, while semantic transformations are prevalent in many real-world integration tasks, as witnessed in workloads of the data curation tool Tamr [21] and other data vendor companies, to the best of our knowledge, no automatic system or tool is available to cover this class of transformations. Semantic transformations cannot be computed by solely looking at the input values, for example, by applying a formula or a string operation. Rather, the required transformations are often found in a mapping table that is either explicitly available to the application (e.g., as a dimension table in a data warehouse) or is hidden behind a transformation service or a Web form.

While collecting adequate reference data resources for a specific transformation task is doable, the process requires tedious crawling and curation exercises that cannot scale to a large number of transformation tasks covering various domains and topics. Indeed, the subject matter expert in a company who is interested in converting its data oftentimes does not have access to the reference data or does not have the skills to code the transformation formula. Additionally, when companies need to create dynamic views to explore their data, they may need to dynamically discover the desired transformations that allows them to join multiple sources on a unified attribute. Furthermore, many transformations, such as currency exchange transformations or genome to coordinates mappings change over time, making previously acquired reference datasets obsolete. Finding the right transformation resources and applying them to a data transformation task is the main goal of this paper.

The Web contains many resources, such as tables, services, and forms, which can be used to perform syntactic and semantic transformations. On the one hand, Web tables are better suited for transforming categorical data [22], such as *product to brand* or *city to state*. Categorical data has finite domains and often has a small number of possible values, hence the required transformations are more likely to be found in tables. On the other hand, Web forms and Web services provide functions for numerical transformations, such as unit conversion and currency conversion, and non-numerical transformations with very large or infinite domain sizes, such as *geo-coordinates to location name* and *ip address to domain.* For simplicity, we limit our definition of Web resources to two main types: Web tables, and Web forms. We assume that a corpus of Web tables has been collected and stored, and that we have a Web form crawling system in place for accessing available Web forms.

This article is published under a Creative Commons Attribution License (http://creativecommons.org/licenses/by/3.0/), which permits distribution and reproduction in any medium as well allowing derivative works, provided that you attribute the original work to the author(s) and CIDR 2015.
7th Biennial Conference on Innovative Data Systems Research (CIDR ’15) January 4-7, 2015, Asilomar, California, USA.

We assume that a data steward has identified a transformation task, and specifies it to `DataXFormer`, as a variable-size collection of examples, giving the entire input X and some examples of the desired output Y. The following are examples of a *airport code to city name* transformation: $\{(LHR, London), (ORD, Chicago), (CDG, Paris)\}$. As a general rule, the steward will have a column in a data set that he wishes transformed (such as *airport code*), from which he has chosen example elements (X, Y). The goal is to find the transformations (values of Y) for the remaining values. In most cases, the user provides the names of the input and output values for the desired transformation, which we denote by I_X and I_Y, but this is not necessary.

We say that a Web resource "covers" a transformation if it contains more than a user-defined threshold τ of the example cases. A naïve approach would be to test every Web form and Web table for coverage, and then use the resource to fill in as many missing values as possible. When multiple resources cover a transformation and provide conflicting values, a resolution scheme would be required to choose the best result. Obviously this solution is linear in the number of Web forms and Web tables, and hence cannot scale. A scalable solution to the above problem requires:

1. indexing a large corpus of Web tables to support efficient retrieval of Web resources that cover a given transformation task;
2. developing a platform that judiciously uses the two types of resources (tables and forms) given their very different access and response time characteristics;
3. effectively involving a crowd of experts (expert sourcing) to guide the transformation tasks, where necessary; and
4. consolidating the possibly inconsistent mapping results from the relevant transformation resources.

To this end, we present `DataXFormer`, an interactive system that leverages Web tables and Web forms to perform transformations based on the given transformation examples. In addition, `DataXFormer` provides users with direct feedback on their transformation results, and allows them to add more examples or to choose from possible transformation alternatives. In particular, we make the following contributions:

- We present an inductive filter-refine approach for indexing and retrieving a large corpus of Web tables to efficiently answer transformation queries.
- We show how relevant Web forms can be retrieved from the Web and (semi) automatically wrapped using the input transformation examples and human experts.
- We present an analysis of the coverage of our system based on 50 real-world transformations.

An online version of `DataXFormer` can be found on the project's website at `http://www.dataxformer.org`

2. DataXFormer OVERVIEW

The architecture of `DataXFormer` is illustrated in Figure 1. It 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 in Section 1 that transforms airport codes to city names. `DataXFormer` converts the user query into an internal form for each of the retrieval components, which return candidate Web tables and Web forms, respectively. While the table retrieval component works on top of a local repository, the form retrieval component uses the entire Web to find relevant Web forms. We assume that the user provides as input a set X of n values and a set E of m example pairs, $E = \{(x_i, y_i) \mid x_i \in X, 1 \leq i \leq m\}$. The goal is to discover the missing Y values and output a solution F, $F = \{(x_i, y_i) \mid x_i \in X, 1 \leq i \leq n\}$.

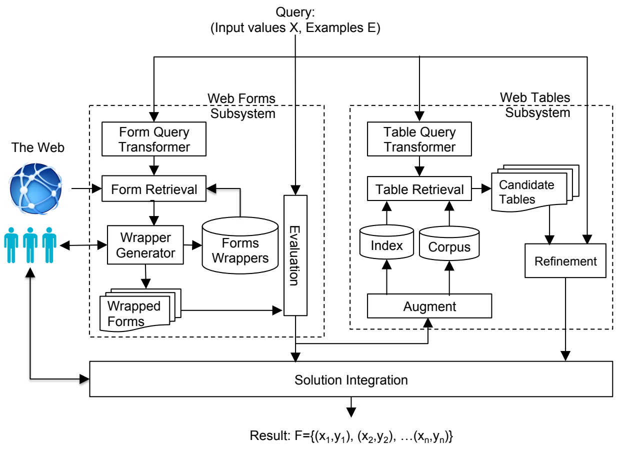


Figure 1: `DataXFormer` architecture

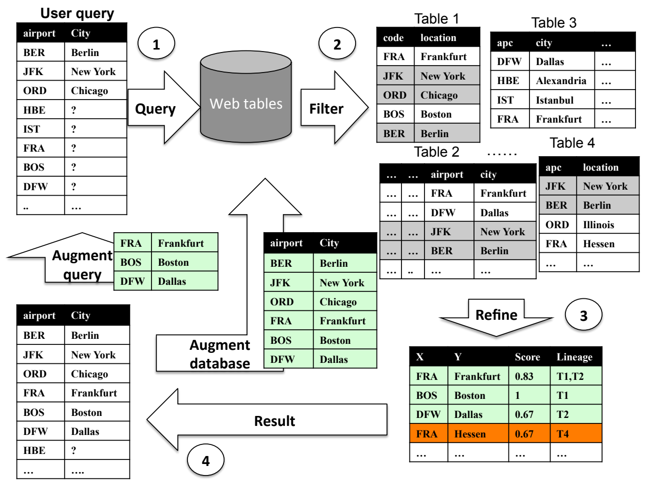
Using an inverted index, the table retrieval algorithm retrieves a set of relevant candidate tables for the given user query. Candidate results are further analyzed by the *refinement* component, which verifies the coverage of the candidate tables with respect to the query examples. If the coverage is above the user-defined threshold, `DataXFormer` extracts the rest of the required transformation values. In the case of web forms, we first use a Web search engine to retrieve relevant URLs. By examining the results of the web search, we identify candidate Web forms. Then, for each candidate form, we generate a "wrapper" that will allow `DataXFormer` to query the form and to obtain the transformation values. Candidate Web forms are then queried using the examples present in the user query. Those with sufficient coverage are then invoked with the remaining input values.

In a final step, the *solution integration* component consolidates and ranks multiple transformation solutions for a given query, as obtained from the two subsystems, and outputs the desired transformation. When automatic reconciliation of values cannot be performed, the expert sourcing system is invoked.

`DataXFormer` dispatches the transformation query simultaneously to both transformation engines. While the two subsystems differ in their respective retrieval interface, response time, and coverage, we are currently taking the straightforward approach of engaging both systems simultaneously. As a future extension, we plan to develop a new scheduler that will take into account information such as query characteristics, run-time constraints, and coverage statistics to develop integration strategies that would maximize query coverage while minimizing the cost incurred from finding and wrapping Web forms.

3. USING WEB TABLES

As discussed in Section 1, a table T is *relevant* to a transformation query if it contains at least τ of the examples provided in that query, where τ is a predefined threshold. Based on the number of contained examples, `DataXFormer` assigns scores to the tables. These scores weigh in when reconciling conflicting transformations of different tables to ensure high quality results, as we show in Section 3.2.



User query

airport	City
BER	Berlin
JFK	New York
ORD	Chicago
HBE	?
IST	?
FRA	?
BOS	?
DFW	?
..	...

Table 1

code	location
FRA	Frankfurt
JFK	New York
ORD	Chicago
BOS	Boston
BER	Berlin

Table 3

apc	city	...
DFW	Dallas	...
HBE	Alexandria	...
IST	Istanbul	...
FRA	Frankfurt	...

Table 2

...	...	airport	city
...	...	FRA	Frankfurt
...	...	DFW	Dallas
...	...	JFK	New York
...	...	BER	Berlin
...	..	...	...

Table 4

apc	location
JFK	New York
BER	Berlin
ORD	Illinois
FRA	Hessen
...	...

X	Y	Score	Lineage
FRA	Frankfurt	0.83	T1,T2
BOS	Boston	1	T1
DFW	Dallas	0.67	T2
FRA	Hessen	0.67	T4
...	...	...	...

FRA	Frankfurt
BOS	Boston
DFW	Dallas

airport	City
BER	Berlin
JFK	New York
ORD	Chicago
FRA	Frankfurt
BOS	Boston
DFW	Dallas

airport	City
BER	Berlin
JFK	New York
ORD	Chicago
FRA	Frankfurt
BOS	Boston
DFW	Dallas
HBE	?
...	....

Figure 2: Workflow of `DataXFormer`'s filter-refine approach

Scanning every table in the corpus for the given examples can be prohibitively expensive. To mitigate such a cost, `DataXFormer` uses a filter-refine approach as illustrated by multiple steps in Figure 2. In Step 1, a query is submitted to the Web table repository, which maintains millions of n-ary relations. Step 2 corresponds to the *filter* phase. Here, `DataXFormer` locates a candidate set of relevant tables. To enable the candidate generation, we need to tackle two major challenges:

1. We have to store and index the tables in a way that allows to retrieve them by examples, without accessing too many irrelevant tables. At the same time, we want to preserve helpful structure information of the tables, such as column and row correspondence of values.

2. It is necessary to reach tables that are relevant to a transformation task but do not contain the initial given examples.

In Figure 2, Tables 1,2, and 4 cover $\tau = 2$ examples and are therefore candidate tables. The *filter* phase significantly affects the recall of the results. For example, a coarse-grained index will return many false positives while a very restrictive query can result in low recall because it misses relevant tables.

In the *refine* phase (Step 3 in Figure 2), we validate the row correspondences of found examples and compute scores of found transformations and tables that contain them. We follow an iterative and inductive approach. An iteration refers to one pass of filter and refine. Since the initial examples might not be representative or might have a low recall, we use newly discovered mappings as examples in subsequent iterations. When the iterations converge, the final transformation results are presented to the user (Step 4 in Figure 2). Furthermore, if the user is satisfied with the result or any subset of the results, she can trigger the system to store the results as a new table with a higher initial confidence score in the database.

We elaborate on how `DataXFormer` handles the filter approach by showing the index design and query formation in Section 3.1 and on how the refine phase of `DataXFormer` assigns scores and reconciles conflicting results in Section 3.2.

3.1 Candidate generation: The filter phase

The simplest way to identify the tables that support a transformation is to use an inverted index that maps example entries to the indexed tables and columns. Since we are interested in finding only the two relevant columns, a column-based solution naturally suits our problem. We adopt a two-phase approach: (i) identify relevant tables and then load their content to validate the examples, and (ii) produce transformations for the remaining values. We implemented this scenario in two different storage systems: a relational column-store database with a star schema and a document index where each column is represented by a document.

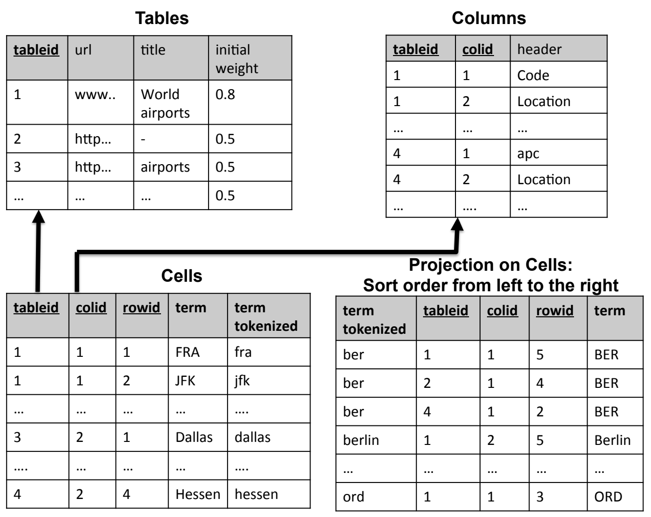
Web table storage and indexing.

Since Web tables are heterogeneous, differ in schema and size, and some even lack column labels, we store the tables within a universal main table (relation *Cells* in Figure 3) where 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. 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 can be alleviated by compression, which is provided by most modern column-stores.

In the current prototype, we store our 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 relation *Cells* that is sorted on the tokenized values.

Besides table content, we store other metadata such as the number of rows in a table in order to easily prune tables with fewer rows than the required number of covered examples. Additional dimension tables are maintained (Figure 3). Relation *Tables* stores meta-data of tables, such as the URL where the table came from, the table title and an initial table weight that may vary depending on the authoritativeness of the table. The initial weight influences the confidence score that is computed later to rate transformation results. We will refer to the score computation, when talking about reconciliation of conflicting results in Section 3.2. Although the table title and the URI are not being used in the current prototype, we maintain them for future work that might incorporate the provenance of stored tables. Relation *Columns* stores columns meta data, such as column headers.

A second option for implementing the inverted index is to



Tables

tableid	url	title	initial weight
1	www..	World airports	0.8
2	http…	-	0.5
3	http…	airports	0.5
…	…	…	0.5

Columns

tableid	colid	header
1	1	Code
1	2	Location
…	…	…
4	1	apc
4	2	Location
…	….	…

Cells

tableid	colid	rowid	term	term tokenized
1	1	1	FRA	fra
1	1	2	JFK	jfk
…	…	…	…	….
3	2	1	Dallas	dallas
….	…	…	…	….
4	2	4	Hessen	hessen

Projection on Cells: Sort order from left to the right

term tokenized	tableid	colid	rowid	term
ber	1	1	5	BER
ber	2	1	4	BER
ber	4	1	2	BER
berlin	1	2	5	Berlin
…	…	…	…	…
ord	1	1	3	ORD

Figure 3: Schema for storing Web tables in a column-store in correspondance with the example in Figure 2

use a document index and to treat the problem as a keyword search problem over documents. For the document index, the simplest way to create index entries from the corpus of tables is to store each column of a table as a separate text document. This approach, while achieving high recall, obscures important structure information that can be used to prune irrelevant tables. However, since values are more likely to be repeated in the same column, data compression gives us the added benefit of a smaller index size. In Section 6, we report on the performance of both storage systems.

Querying Web tables.

We query for column pairs that contain at least τ of the given examples. To maximize coverage, we tokenize and stem every value x and y from the input. Using the *Cells* relation described earlier, `DataXFormer` filters all relevant column pairs with a single SQL query:

```
SELECT col1.tableid, col1.colid, col2.colid
FROM
(SELECT tableid, colid
 FROM Cells
 WHERE term_tokenized IN (<x1>,<x2>,..., <xm>)
 GROUP BY tableid, colid
 HAVING COUNT(DISTINCT term_tokenized) >= tau)
 AS col1,

(SELECT tableid, colid
 FROM Cells
 WHERE term_tokenized IN (<y1>,<y2>,..., <ym>)
 GROUP BY tableid, colid
 HAVING COUNT(DISTINCT term_tokenized) >= tau)
 AS col2

WHERE col1.tableid = col2.tableid
AND col1.colid <> col2.colid
```

The query joins two subqueries, one to find the columns that contain the X values and another to identify the columns that contain the Y values, where a column is uniquely identified using the table and the column IDs. By comparing the table IDs, `DataXFormer` retrieves only column pairs that appear in the same table. We also make sure that a result entry consists of two distinct columns. The result of the query is a set of triples, each comprising a table ID and two column IDs. We need the combination of table and column IDs to identify a column because a column ID is unique only within a specific Web table.

Note that query processing becomes increasingly costly as the number of examples in the query increases. This issue becomes relevant as we follow an inductive approach. After a *filter-refine* iteration, we use retrieved transformations as examples to find additional tables that eventually cover missing X values. Since we cannot include an arbitrarily large number of examples, we must limit the set of examples. `DataXFormer` chooses examples based on confidence scores that are computed and updated after each filter-refine iteration. We describe the reconciliation process and how these confidence scores are computed and used in Section 3.2.

When a document store is used instead of a RDBMS, `DataXFormer` has to formulate and submit a keyword query for each example combination of size τ to find relevant columns "documents". This approach results into $2 \cdot \binom{k}{\tau}$ queries for k given examples X and Y. The columns matching each query are then joined on the table identifier.

3.2 The Refine Phase

In the *refine* phase, we load the content of each candidate column pair and check the row correspondence between the values. Note that the candidate generation does not ensure the transformation examples to be aligned in the corresponding rows. If τ examples still match after considering the row correspondence, `DataXFormer` collects all transformation results that are provided by the corresponding table.

The retrieved tables might provide conflicting answers, i.e., returning different Y values for the same X value. For example, in Figure 2, the airport code "FRA" has been assigned to two different values. A naïve approach to resolve such conflicts is to apply majority vote. However, we would like to take into account the authoritativeness of tables. For example, while *(JFK, New York)* might appear in more tables of the database, a table from a more reliable source might provide *(JFK, New York City)*, which is more accurate. Therefore, it is necessary to score tables according to their authoritativeness as well as their coverage of examples with a confidences score. Furthermore, as we incorporate results of previous iterations as examples, we also need a confidence score for those examples. We therefore adopt an iterative expectation-maximization (EM) approach [11] that incorporates confidence scores. The confidence score of each table (i.e., the probability that an answer it provides is correct) is estimated based on the current belief about the probability of the answers. Initially each table is assigned with a confidence score based on the number of user examples it covers. The score of the table is weighted with its initial weight, which was assigned by experts and stored in relation $Tables$. The answer scores are updated based on the newly computed scores, and the process is repeated until convergence is reached (i.e., the sum of all score changes is below a very small value ϵ). In the end, for each $x \in X$, the scores of possible answers form a probability distribution.

Algorithm 1 Expectation-Maximization

```
Input: A set of initial examples $E = \{(x, y), \ldots\}$, a set of required values $X$
Output: Scored answers
 1: $answers \leftarrow E$ //Represent the input query as an absolute reference
 2: $tables \leftarrow E$
 3: $finishedQuerying \leftarrow false$
 4: $oldAnswers = answers$
 5: repeat
 6:   if not finishedQuerying then
 7:     $tables \leftarrow$ QueryForTables($answers$)
 8:     for all $table \in tables$ do
 9:       for all $answer(x, y) \in table$ do
10:         if $x \in X$ then
11:           UpdateLineage($table, answer$)
12:           $answers$.Add($x, y$)
13:     if not new $X$ was covered by tables then
14:       $finishedQuerying \leftarrow true$
15:   UpdateTableScores($answers, tables$) //maximization step
16:   UpdateAnswerScores($answers, tables$) //Expectation step
17:   $\Delta scores = \sum |answers.score(x, y) - oldAnswers.score(x, y)|$
18:   oldAnswers = answers;
19: until $finishedQuerying \wedge \Delta scores < \epsilon$
```

Currently, DataXFormer reports the highest scoring value as the answer.

Algorithm 1 describes the reconciliation process embedded within the overall workflow of DataXFormer's filter-refine iterations. In each iteration, DataXFormer queries for tables using the new weighted examples (line 7), until no more values in X can be covered (lines 6 and 14). The algorithm then continues the iterative process without querying for new tables, until the scores converge (line 19).

In line 15, DataXFormer implements the maximization step by updating table scores (estimated error-rates) based on the current belief in the answers (answer scores). In the beginning, the only examples present are the ones given by the user. Initial scores are assigned based on the percentage of examples that were covered by a table. In each iteration, if new unseen tables were found in the query, the lineage of the newly found tables and the answers (transformations) they provide are recorded for later EM calculations. When updating table or answer scores, it is necessary to be able to identify which tables contained which answers. In the expectation step of every iteration, the scores (i.e. probabilities) of the discovered answers are updated. DataXFormer converges when the total (absolute) change in answer scores is below a small value ϵ. The expectation step and the maximization step are illustrated in more detail in Appendix A.

Using an EM approach allows for seamless integration of feedback, e.g., through expert sourcing, by adjusting the scores and iterating until a new convergence point is reached. Furthermore, it is easy to assign higher initial weights to more authoritative tables that were provided by expert users, or to edit the examples on the fly in an interactive session, where the system recomputes the results after the user marks correct and incorrect answers. The superiority of EM over majority vote for our use case is demonstrated in Section 6.3.3.

4. USING WEB FORMS

As with Web tables, we assume a form is relevant to a query 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 crawl for relevant ones from the Web; and (2) a new Web form appears as a black box, and an invocation strategy (i.e., wrapper) has to be developed to use the form to produce the desired transformations. It has been shown [4] that both tasks are very hard, even with human intervention. In the following, we show how DataXFormer automatically retrieves forms from the Web and wraps them for automated invocation.

4.1 Web Form Retrieval

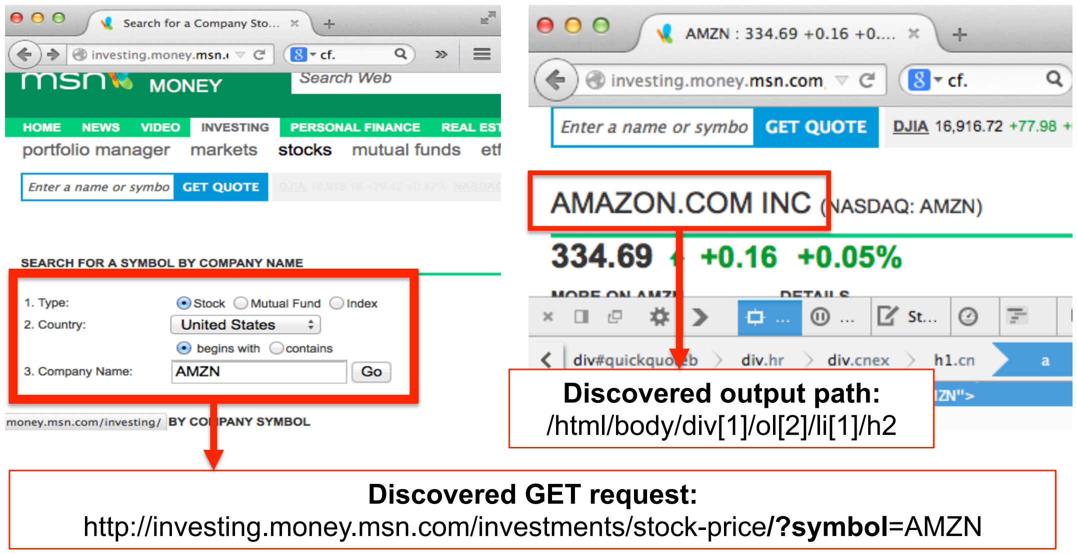
DataXFormer dynamically searches for relevant forms from the Web by issuing search queries (on existing engines) with the attribute names I_X and I_Y. As part of future work, we plan to investigate techniques that use the example X and Y values in E to directly identify relevant forms. During this process, DataXFormer maintains a repository of Web forms that have been successfully wrapped and previously used to answer transformation queries. Each Web form is stored as a document that contains the attribute names I_X and I_Y, frequent terms from the form's Web page, and examples from previous transformation tasks. In addition to querying the Web, DataXFormer queries this repository for candidate matches along with their corresponding wrappers.

In our preliminary experimental results (Section 6), we noticed that the keyword query I_X `to` I_Y has a high success rate for finding a page containing a transformation Web form in its top results. A page contains a Web form if it contains an HTML form header and input fields.

Since the number of Web forms is far less than the number of regular pages retrieved by a search, DataXFormer issues multiple keyword queries and filters the results coming from the underlying search engine to find Web forms. Example keyword queries include terms such as "convert", "detect", or "lookup" added to the column names.

4.2 Wrapping Web Forms

To be able to wrap Web forms, we have to simulate a Web browser and probe the forms by using the given example values $(x, y) \in E$ 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 form. Depending on the invocation method of the Web form, which can be either an HTTP method or Javascript code, we decide on how to identify the input fields. Current wrapper generation approaches [4, 19] invoke a Web 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 of DataXFormer, we apply some heuristics to reduce the number of combinations. To invoke HTTP-based forms, we send the HTTP request by probing only each input text field within the form. Some forms, especially those that provide transformations in both directions, also contain *select* fields or *radio buttons*, where the appropriate option has to be selected. Instead of trying all possible options, we set the *select* and *radio buttons* to the options that match the attribute names I_X and I_Y. If the option contains I_X as well as I_Y, e.g., the option text is "km to miles", we use the order of the attribute names to identify the transformation direction. In the worst case, we need to try each option in a brute force manner.


Search for a Company Sto...
investing.money.msn.
msn MONEY
HOME NEWS VIDEO INVESTING PERSONAL FINANCE REAL EST
portfolio manager markets stocks mutual funds etf
Enter a name or symbo GET QUOTE
SEARCH FOR A SYMBOL BY COMPANY NAME
1. Type: Stock Mutual Fund Index
2. Country: United States
begins with contains
3. Company Name: AMZN Go
money.msn.com/investing/ BY COMPANY SYMBOL
AMZN : 334.69 +0.16 +0....
investing.money.msn.com
Enter a name or symbo GET QUOTE DJIA 16,916.72 +77.98
AMAZON.COM INC (NASDAQ: AMZN)
334.69 +0.16 +0.05%
div#quickquoteb div.hr div.cnex h1.cn a

Discovered output path: /html/body/div[1]/ol[2]/li[1]/h2

Discovered GET request: http://investing.money.msn.com/investments/stock-price/**?symbol**=AMZN

Figure 4: The wrapper for this Web form consists of a GET URI, the request parameter `?symbol` and the output path

When dealing with `JavaScript`, `DataXFormer` has to identify the relevant `JavaScript` transformation function and its parameters. Parsing and simulating all possible `JavaScript` functions on a Web page is a tedious task [19]. In the current prototype, we identify all `JavaScript` functions that are executed by an *onClick* event. We then parse these functions' parameters to locate those that match the names of the input columns I_X and I_Y. The remaining parameters are set to their default values to get a valid invocation of the function. In both cases, once a request succeeds, we scan the returned HTML page for the y value and fetch the absolute XPath, which includes all HTML tags from the root of an HTML document to the HTML tag that contains the output value. Multiple example pairs from E are used to avoid accidental matching. For example, for a given example pair $(x, y) \in E$, we might find the y value in multiple fields of the result page, especially if y is a number that is accidentally found in a field that is unrelated to the request. Those fields can be easily dismissed by probing more examples from E.

Figure 4 shows a candidate Web form that has been retrieved for transforming stock names to companies, e.g., `AMZN` to `AMAZON.COM INC`. The screenshot on the left-hand side shows the form's input fields that comprise various radio buttons, a selection field, an input field, and a button. By analyzing the HTML code, `DataXFormer` discovers that the form can be invoked by a GET request URI with the relevant parameter `?symbol` that has to be set to our input value `AMZN`. Submitting the GET request returns the page that is illustrated on the right of Figure 4. The desired output `AMAZON.COM INC` appears on the page. `DataXFormer` discovers the XPath of the output field. Knowing the path, `DataXFormer` can identify the transformation for any subsequent stock symbol.

Once `DataXFormer` successfully wraps a Web form, we create a new entry in our Web form repository by storing its URL, invocation method, and the canonical XPath of the relevant fields along with previously mentioned meta-data.

5. TRANSFORMATIONS WITH THE CROWD

`DataXFormer` involves tackling multiple challenging tasks which can be improved through human intervention. Human experts or users can help generate better search queries, validate candidate tables or forms, devise more accurate wrappers, and reconcile inconsistencies among the results from multiple transformation resources.

However, using the crowd comes with the well-known problems associated with crowdsourcing tasks, such as deciding on the right incentives and controlling the quality of the results [16]. This is especially true in cases where the human agent is not provided with choices, but rather is required to produce content. `DataXFormer` is geared more towards enterprise environments and hence assume the availability of an expert sourcing system. By the nature of internal enterprise experts, this will avoid some of the issues with conventional crowdsourcing. Involving experts requires translating a given transformation query into several meaningful crowdsourcing units, often referred to as human intelligence tasks (HITs). Forming effective HITs is a main crowdsourcing challenge. In `DataXFormer`, we use straightforward HIT forming approaches depending on the task. We leave more sophisticated HIT generation for future work.

Results Generation and Consolidation `DataXFormer` consolidates retrieved transformation results by using an expectation maximization model. However, when these rules fail to resolve inconsistencies or the user is not satisfied with the consolidated results, `DataXFormer` provides the experts with HITs comprising the contradicting transformations and the domain identifiers I_X and I_Y, and asks the experts to choose the correct transformation (cf. Figure 1). Simple majority voting is used to resolve conflicts and the votes are further used to rate the confidence of the transformations.

If the Web tables and Web forms subsystems fail to provide a transformation for some values in X, the crowd can also be directly polled to produce the answers. A production task looks like an analogy game: the worker is provided with some examples for the transformation and is required to fill in the missing values. Again, a HIT must not contain more than a specified number of tuples to be perceivable by a worker. If very few examples are available, many HITs will fail to get meaningful answers, because workers will not be able to curate inconsistent transformation results. To tackle this problem, we follow the inductive approach we used in Section 3, where discovered mappings are used to form the examples of new HITs.

Given the example pairs and the column identifiers I_X and I_Y, an expert can also be asked to find a Web page that contains a relevant Web form. As discussed in Section 4, the task of wrapping Web forms requires parsing Web forms and identifying the semantics of the various components, e.g., using evidence from the field labels. Experts can easily provide hints about the input and output fields in a given Web form. For example, given a form `URL`, the agent can be asked to reply back with the name (and mouse click coordinates) of the field that receives the X values and the name (and coordinates) of the field that produces the required Y values. More sophisticated tasks include asking the experts to identify the transformation function in the `JavaScript` in a given Web form.

6. CASE STUDY

In this section, we present our preliminary results in evaluating the effectiveness of `DataXFormer` using a workload of 50 real-world queries collected from a data curation company and data scientists. We first describe our experimental setup, then we show how `DataXFormer` was able to cover 82% of the workload by using the techniques presented in this paper. Finally, we examine the effects of the different parameters and compare the performance of the two proposed solutions for indexing and querying the tables.

6.1 Experimental Setup

For Web tables, we used the Dresden Web Tables Corpus provided by Eberius et al.[1] This corpus comprises 112 million Web tables extracted from the large Common Crawl corpus, which contains about 4 billion Web pages, with a large percentage missing column or table headers. We stored and indexed the tables using both options described in Section 3.1: the Vertica DBMS [18] and a document index provided by the Lucene search engine[2].

The size of the stored tables and projections is 350 GB using the Vertica DBMS. In comparison, the index size of the Lucene document index is 64 GB, while the corpus is 202 GB.

Lucene supports block indexing, which allows grouping related documents together. As we treat columns as documents, we group columns from the same table in a single block, effectively pre-materializing the join on table identifiers.

To collect a test workload, we asked data scientists and employees from a data curation company with commercial customers to provide examples of common transformation queries they face in real-world applications. In total, we collected 50 queries that comprise both syntactic and semantic transformations (See Appendix B for the full list).

	Form found and wrapped	Found but not wrapped	No forms found	
Covered by tables	12	5	12	29
Not covered by tables	12	5	4	21
	24	10	16	50

Table 1: `DataXFormer` coverage of transformation queries: 29 queries were covered by Web tables and 24 by Web forms

To evaluate the effectiveness of our system, we consider two metrics: (i) the coverage and (ii) the quality of individual transformations. The coverage refers to the fraction of transformation queries where `DataXFormer` is able to find relevant tables or Web forms. The transformation quality assesses the correctness and completeness of a transformation task in terms of precision and recall. In our experiments, we manually obtained example input for all covered queries to measure the recall. For evaluating the precision of a transformation, it is necessary to know the ground truth. Therefore, we additionally obtained the ground truth for a subset of the queries to measure the precision. The ground truth of a query contains the complete domain and range of a transformation, e.g., all airport codes and corresponding cities.

6.2 Query Coverage and Recall

For each of the 50 queries, we manually generated 10 input values (total number of 500 input values) and checked whether `DataXFormer` returned any result. We manually judged whether the result was correct or not.

The union of the Web tables and Web forms subsystem covers 41 (82%) queries considering that 12 queries could be answered through both. For 9 queries we neither found a wrappable Web form nor a Web table that contained any of the input values. Table 1 shows the coverage details of the two subsystems (Web tables and Web forms). 29 of the queries could be answered using Web tables. Those queries include mapping categorical data, such as *city to country*, *ISBN to publisher*, and *car model to car brand*. At the same time, for 34 of the queries, `DataXFormer` found relevant Web forms through the Web search engine. The average ranking position of the Web page containing the relevant Web form was 3.2. For 24 out of those 34 queries, `DataXFormer` was able to wrap the form and to produce the correct transformations. The Web forms, which could not be wrapped, either maintained no HTML structure for effectively parsing the `JavaScript` functions or required submitting specific header information with the HTTP request that could not be automatically identified. These are our candidates for human-generated wrappers through crowdsourcing.

On average `DataXFormer` achieved 81.3% recall for the 41 covered queries. We present the recall for each specific query in Appendix B. Queries that were covered by Web forms usually resulted in 100% recall. High recall was also achieved for queries that could be covered by a single table from the Web table corpus,(e.g., *country to country code*). Note that some of the queries represented classes of queries. For example, we found and wrapped a Web form for transforming *USD to EUR*. In general, our wrapper is able to wrap the

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

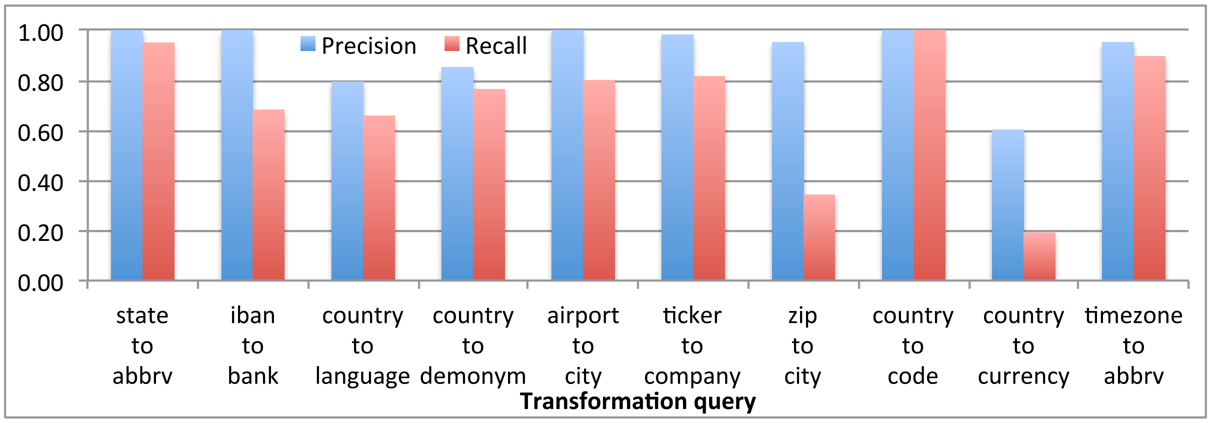


Figure 5: Precision and Recall of `DataXFormer` **for all given queries**

same Web form for all other possible currency conversion transformations that were not part of our 50 queries. The same applies to the class of unit conversion transformations, where we only included *Fahrenheit to Celsius* in our set of queries.

6.3 Transformation Quality

The transformation quality of the Web table component depends on a set of parameters and algorithmic features that we analyze here in more detail. We assess the quality performance of our Web table system by measuring the precision and recall of the results for different set of initial examples, different thresholds, and different reconciliation approaches. We show that `DataXFormer`'s performance is not affected by the popularity of example terms, but rather by the threshold τ and the scoring propagation system in the refine phase.

To additionally check the precision of our algorithm, we have to compare `DataXFormer`'s results to ground truth. For this purpose, we manually obtained the complete directory of input values and transformation results (e.g., airport codes and corresponding cities) for 10 of the queries that were covered by the Web table subsystem. Since, our current corpus of Web tables might have limitations, we consider only those value pairs X and Y from the ground truth that appear in at least one table with more than two entries. Thus, we are able to assess the actual effectiveness of `DataXFormer`'s algorithm.

Figure 5 illustrates the precision and recall of `DataXFormer` for the analyzed queries. For each query, we provided 5 randomly chosen examples. All queries were run with $\tau = 2$. For seven queries, `DataXFormer` achieves more than 90% precision. For the remaining three cases, *country to currency*, *country to languages*, and *country to demonyms*, we observe that recall is similarly low. Looking carefully at these cases, we observed that the refine phase picked the wrong transformations, which caused both precision and recall to decrease. For the specific cases of languages and demonyms [3], the wrong reconciliation stems from ambiguity provided by the examples, e.g., French refers to the language in France as well as to its residents. In the case of the query *country to currency*, we face low precision and recall because of different representation of currency values (e.g., "Qatari riyals" vs "rial"), which would require more robust matching algorithms. For the queries, such as *zipcode to city* or *iban to bank*, where high precision was achieved, the low recall is caused by the fragmentation of the data among many tables, which impedes their reachability based on the given examples. In the following, we analyze the effect of variation in the number and type of examples on the `DataXFormer`'s effectiveness.

[3] A demonym denotes the resident of a country.

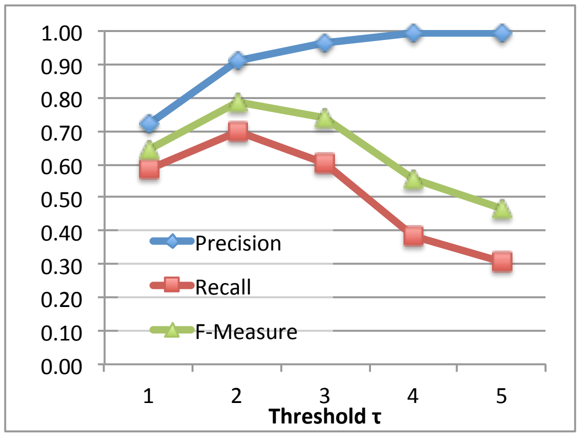


Figure 6: Average Precision and Recall based on different τ configurations

6.3.1 *Effect of parameter τ*

A key parameter for filtering Web tables and Web forms is τ. A table or form supports a transformation if it contains at least *tau* of the examples. Figure 6 shows the effect of τ on the average precision and recall of our ten queries. For this experiment, we fixed the number of initial examples to 5. Therefore, we can only consider τ values between 1 and 5. The figure shows that the precision increases with τ. However, there is a trade-off with regard to recall. The higher τ, the more tables are filtered out resulting in a low recall, e.g., for $\tau = 5$ `DataXFormer` achieves only about 30% recall. In fact, 7 out of the 10 queries had a recall values less than 8% for $\tau = 5$. Only the queries *states to abbrevia-*

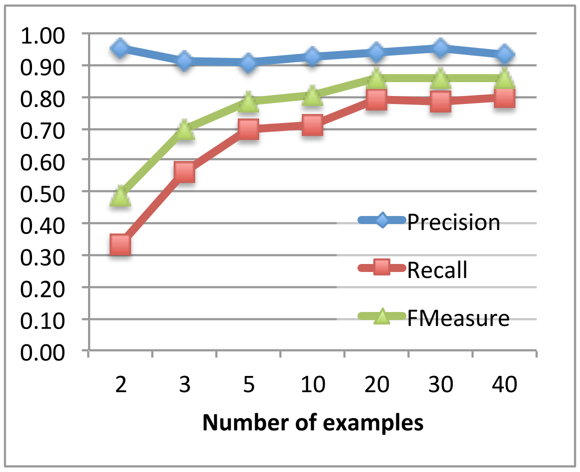


Figure 7: Average Precision and Recall based on the number of examples

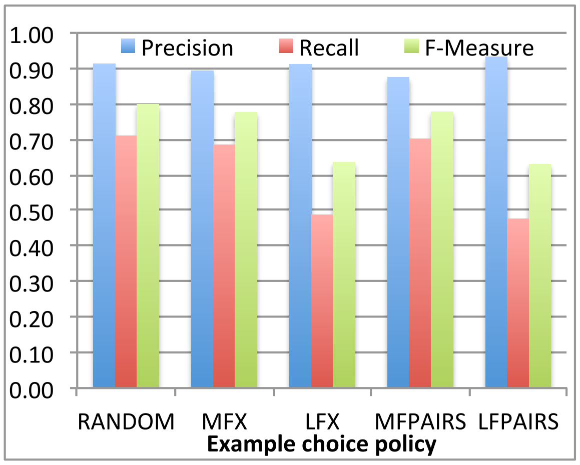


Figure 8: Average Precision and Recall for different choices of initial examples

tion, *country to country code*, and *timezone to abbreviation* achieved recall above 90% because the results for each all appeared in a single table. In Appendix B, we report for all queries whether there is a single table that contains all the requested transformations. The optimal value is at $\tau = 2$, where recall and F-measure (the harmonic mean of recall and precision) are the highest. The precision is only slightly lower than in experiments with $\tau > 2$. Therefore, we configured $\tau = 2$ for all other experiments to achieve the most promising results. Note that we did not achieve the highest recall value for $\tau = 1$. This is due to the fact that many unrelated tables with contradicting transformations might have been retrieved. This affects our scoring consolidation system in such a way that it fails to decide for the correct transformations.

6.3.2 *Effect of initial examples*

The number of initial examples for describing a transformation is crucial. The more examples the user provides, the more tables could match the transformation and a higher recall can be achieved. Figure 7 illustrates the average precision and recall across all ten queries with respect to the number of examples. We clearly see that recall increases with the number of examples. Note, for the datapoint corresponding to two examples, most of the queries did not output any transformations, resulting in very low recall. The precision remains continuously stable at around 90%.

In general, it is obvious that the more distinctive an example is to a query, i.e., the example is a good representation of that specific query but no other queries, the better precision results can be achieved. For example, when looking for the transformation *country to demonym*, the example pair *(Brazil, Brazilian)* is a better choice than *(France, French)* since, French not only denotes the citizens of France but also their language. Furthermore, considering the fragmented nature of data in Web tables, the choice of popular examples might yield better recall, because presumably the popular examples will re-occur in many of the relevant tables. Popularity and distinctiveness are hard to assess. Our heuristic to assess the popularity or distinctiveness of an example is to consider its frequency. We deem an example to be more popular than another if it occurs more often in the dataset. On the other hand, we deem an example to be more distinctive if it is less frequent in the dataset. Figure 8 shows precision and recall results of `DataXFormer` using five examples with the most frequent X values (MFX), five examples with the least frequent X values (LFX), five examples with the most frequent pairs X and Y (MFPairs), and five examples with the least frequent pairs X and Y (LFPairs). In addition, we added the results from Figure 7, where the five examples were randomly chosen (RANDOM). Choosing five examples with the least frequent X values or X/Y value pairs yields significantly lower recall than the examples with frequent X or X/Y pairs, while the precision is only slightly higher. We can conclude that low frequency does not represent distinctiveness of examples. Interestingly, our random choice of examples beats all four heuristics, showing that the frequency of the examples is not a good heuristic for assessing the appropriateness of initial examples.

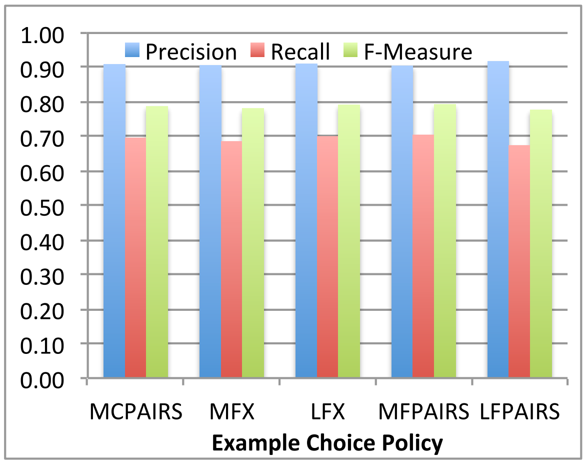


Figure 9: Average Precision Recall for different choices of examples in later iterations

In a different experiment, we fixed the initial examples and

Method	Precision	Recall	F-Measure
Majority voting	0.88	0.68	0.77
EM model	**0.91**	**0.70**	**0.79**

Table 2: Majority voting vs. EM propagation system

checked whether the consistent application of the heuristic in later iterations can have any effect on the performance of `DataXFormer`. So instead of choosing all possible examples, we limited the maximum set of generated examples for each iteration to 50. Notice that, for performance issues, there should be an upper bound on the number of examples anyway. Figure 9 illustrates the MFX, LFX, MFPair, and LFPair policies compared to our policy based on confidence of the most certain examples (MCPairs) as discussed in Section 3.2. The results show that all policies yield the same performance. This means that, in contrast to the initial examples, the generated examples after the first iteration widely overlap, no matter which policy we chose. We stick to the policy of choosing the examples (MCPairs) based on the scores derived from the tables because it is the more efficient option. Retrieving the frequency of examples requires additional computation effort.

6.3.3 Scoring Model

We now evaluate the use of the proposed EM model (Section 3.2). A naïve baseline solution to consolidate multiple contradicting transformations would be to apply a majority voting and choose the transformation that is covered by the most number of transformations.

Table 2 illustrates the average precision and recall for both systems using the same set of queries as in Figure 5. The table shows that the EM model improves on majority voting by 3% in precision and 2% in recall. Looking at individual queries, the EM model was always slightly better than majority voting, having the highest gain (20% in precision and 22% in recall) in case of *country to demonyms* where many contradictory solutions appear. EM's superior performance appears when the tables providing the required transformation are dominated by many tables that offer different overlapping transformations. Despite the slight average gain on precision and recall, the EM model suits our purpose well, because it can capture further features, such as initial table weights column header similarity. We will elaborate on augmenting the EM model with these features as part of future work.

6.4 Column store vs. Document Storage

We compared our Vertica storage system with a Lucene index. In the case of Vertica, we can formulate a single SQL query to find all tables that contain τ examples, while for the Lucene index, we need to submit a query for every τ-sized combination of the given k examples. Figure 10 shows the runtime of a query to find matching tables for a number of given *zip to city* examples. For that experiment, we set $\tau = 2$. While both systems have similar runtime for small numbers of examples, the time difference increases significantly for higher numbers of examples. With 50 examples, the query to Vertica is already faster by more than an order of magnitudes than the set of queries sent to the Lucene index. Note that, while the user provides only a handful of

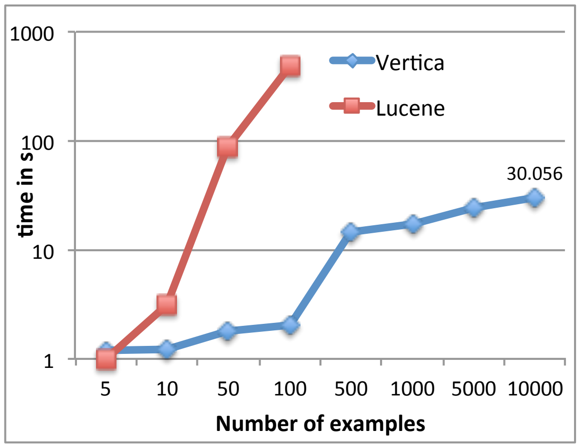


Figure 10: Query time using Vertica vs. Lucene

examples for the first iteration of `DataXFormer`, later iterations have large number of generated examples. For 500 and more examples we stopped the Lucene-based system after 1 hour. Vertica performs well even for very large numbers of examples, e.g., 30 seconds for 10,000 examples. The experiment clearly shows that a DBMS is the more efficient storage option for `DataXFormer`.

7. RELATED WORK

Commercial companies that provide information products, such as Recorded Future[4], apply transformations to raw data to create semantic features. While some transformations are simple, others are complex and involve reference datasets or Web services. However, these transformations are defined manually, while our goal is their automatic discovery and integration.

Academic research on Web tables has usually focused on issues related to search, extraction and integration techniques [6, 7, 10]. Web tables have been also regarded as a large repository for knowledge discovery. For example, InfoGather [22] addresses the problem of entity augmentation by searching for related attributes of given entities. Searching is performed by name, by examples, or simply through automated discovery. The search is expanded by way of a schema matching graph of the underlying Web tables. `DataXFormer` isolates relevant tables from the vast corpus without relying on predefined, structured queries and where the schema is not fully known.

Many techniques have been proposed to provide information retrieval style capabilities over structured databases. Most of the proposed systems [1, 2, 15] focus on efficiently generating candidate networks of joined tuples to form answers to a keyword query, with search terms spanning multiple records across multiple tables. In some of these systems, such as in [3, 13], specialized indices or predefined foreign key-primary key relationships are used to prune the space of candidate results. Others, such as in [5], rely on meta-data information and dependencies among keywords to identify joinable tuples. In `DataXFormer`, we are not interested in tuple networks, but rather in tables with columns covering most of the example transformations and the input values

[4]`http://www.recordedfuture.com`

to be transformed. Since we do not have complete knowledge of the underlying schema, our approach has to depend on instance matching (in the case of Web tables) and on discovering needed meta-data (in the case of Web forms).

A large body of work on wrapper induction and deep Web crawling is highly related to our form retrieval, wrapper discovery and Web form invocation techniques. Wrapper induction mainly focuses on extraction of information from Web pages [8, 9, 14, 20], which can be leveraged to discover the semantics of Web forms components, especially with the aid of human experts.

8. CONCLUSION AND FUTURE WORK

We presented DataXFormer, a system for automated data transformations based on Web tables and Web forms. In particular, we showed how DataXFormer leverages both types of resources for covering syntactic, as well as semantic transformations. In a case-study, we showed that our system covers 82% of transformation queries provided by real users. We plan to conduct an additional extensive user study to confirm this result with more real use cases. In order to do so, we are providing an online version of DataXFormer that accepts user queries, solves them, and solicits feedback at http://www.dataxformer.org

Future work includes the development of more sophisticated integration and synergy between the tables subsystem, forms subsystem, and crowdsourcing. Moreover, the Web form subsystem needs to find and wrap Web forms without the knowledge of the attribute names. Additionally, it is desirable to have a focused crawler that more effectively crawls for possible transformation Web forms in the background. Finally, DataXFormer can be further improved by enabling fuzzy matching for attribute identifiers as well as example values, e.g., allowing to match *US Dollar* with *USD* or *New York* with *New York City.* Currently, DataXFormer incorporates fuzzy matching only rudimentary. The tokenization of cell values allows case- and punctuation-independent matching. To incorporate more sophisticated fuzzy matches based on synonym matching and edit distance, we have to extend our Web table repository with a similarity index or synonym dictionary without significantly impeding the scalablility of DataXFormer. Finally, DataXFormer should also incorporate fuzzy matching scores in its confidence score model.

Beyond the different issues that still need to be tackled there are several possible future paths that we are planning to pursue.

8.1 Multi-columns Input

The techniques discussed thus far allow the discovery of one-to-one transformations. However, functional transformations may depend on more than one argument, therefore the model can be extended from list of values (one values for each example with X) to a list of lists. Consider the following scenario: given a database of basketball players with their personal information, the user wants to encode them with their jersey number. However, this transformation is prone to errors, as there are players with the same name, and a player changes his jersey number when he moves to a different team. Therefore the input should have the player name, the year of interest (or the team), and the date or place of birth. Given such input, the jersey number can be computed in a deterministic fashion.

This process is already possible with alternative approaches, such as Google "Smart Autofill"[5], but only for numerical values in the output (the Y). In fact, these methods are based on mathematical processes that take a matrix as input (multiple columns with their values) and create distributions from which output values can be drawn. In contrast, having a list of values for each example in X raises some technical challenges and opportunities. For example, when querying Web tables, the presence of multiple keywords lead naturally to the need of having joins to related pieces of information that are spread over multiple tables. This leads to the need of discovering tuple networks [1, 2, 15] over Web tables with poor or missing schemas.

Multi-column input has also implications on the Web forms discovery. For example, given a database of Italian residents, they can be encoded with their national number because there is a transformation that takes as input first name, last name, place and date of birth and outputs the national ID of a person. On the one hand, the discovery is easier for our tool, as more labels are available and they can identify the forms of interest more easily. On the other hand, there is a much larger number of combinations of input that must be tested over the form, before discovering the right subset and assignment to fill them.

8.2 Extending Web Resources

We have limited ourselves to Web tables and Web forms. However, both can be extended to increase the recall of the system. To go beyond the current repositories of Web tables, there are open information extraction approaches that create tables from text [6, 12]. These methods can naturally be applied offline to create the tables, but interesting opportunities arise when they are coupled with our application. In fact, our user-provided examples can be seen as seed instances to bootstrap these processes. It would be intriguing to see how these systems can be adapted to achieve interactive response time, given the different input.

Other potential resources to exploit are ontologies and knowledge bases (KBs), such as Yago, DBpedia, and Freebase. For example, a KB can expose a functional relationship between countries and their capitals (i.e., *hasCapital(France,Paris)*), which can then be used for our purposes. Another example is *alias* relationships, such as *alias(New York City, The Big Apple)*. A challenge here is the discovery of the correspondences between the examples and the KBs. In fact, an instance-based algorithm is needed, but the very few examples in Y may return multiple matches, thus requiring humans in the loop to solve ambiguities.

Finally, companies maintain domain-specific tables within their intranet. A very useful extension of DataXFormer would be to be able to connect to those networks in order to be able to collect existing data sources. As company data usually is cleaner than Web tables, we would have to extend our EM model to consider these provenance aspects.

9. REFERENCES

[1] 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.

[5] https://cloud.google.com/prediction/docs/smart_autofill_add_on

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

[3] A. Balmin, V. Hristidis, and Y. Papakonstantinou. Objectrank: Authority-based keyword search in databases. In *VLDB*, pages 564–575, 2004.

[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] S. Bergamaschi, E. Domnori, F. Guerra, R. Trillo Lado, and Y. Velegrakis. Keyword search over relational databases: a metadata approach. In *SIGMOD*, pages 565–576, 2011.

[6] M. Bronzi, V. Crescenzi, P. Merialdo, and P. Papotti. Extraction and integration of partially overlapping web sources. *PVLDB*, 6(10):805–816, 2013.

[7] M. J. Cafarella, A. Halevy, and N. Khoussainova. Data integration for the relational web. *PVLDB*, 2(1):1090–1101, Aug. 2009.

[8] S.-L. Chuang, K. C.-C. Chang, and C. Zhai. Context-aware wrapping: Synchronized data extraction. In *VLDB*, pages 699–710. VLDB Endowment, 2007.

[9] N. Dalvi, R. Kumar, and M. Soliman. Automatic wrappers for large scale web extraction. *PVLDB*, 4(4):219–230, 2011.

[10] A. Das Sarma, L. Fang, N. Gupta, A. Halevy, H. Lee, F. Wu, R. Xin, and C. Yu. Finding related tables. In *SIGMOD*, pages 817–828, New York, NY, USA, 2012.

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

[12] O. Etzioni, A. Fader, J. Christensen, S. Soderland, and Mausam. Open information extraction: The second generation. In *IJCAI*, pages 3–10, 2011.

[13] J. Feng, G. Li, and J. Wang. Finding top-k answers in keyword search over relational databases using tuple units. *TKDE*, 23(12):1781–1794, 2011.

[14] B. He, Z. Zhang, and K. C. Chang. Towards building a metaquerier: Extracting and matching web query interfaces. In *ICDE*, pages 1098–1099, 2005.

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

[16] P. G. Ipeirotis, F. Provost, and J. Wang. Quality management on amazon mechanical turk. In *HCOMP*, pages 64–67, 2010.

[17] 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.

[18] A. Lamb, M. Fuller, R. Varadarajan, N. Tran, B. Vandiver, L. Doshi, and C. Bear. The vertica analytic database: C-store 7 years later. *PVLDB*, 5(12):1790–1801, Aug. 2012.

[19] 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.

[20] A. Parameswaran, N. Dalvi, H. Garcia-Molina, and R. Rastogi. Optimal schemes for robust web extraction. *PVLDB*, 4(11), 2011.

[21] 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.

[22] 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, 2012.

Algorithm 2 UpdateTableScores

```
Input: $answers$, $tables$
Output: estimated table scores
 1: for all $table \in tables$ do
 2:     $good \leftarrow 0$
 3:     $total \leftarrow 0$
 4:     $coveredXs \leftarrow \{\}$ //holds example x's appearing in the table
 5:     for all $answer(x, y) \in table$ do
 6:         $coveredXs \leftarrow coveredXs \cup \{x\}$
 7:         $score \leftarrow$GetScore$(x, y)$
 8:         if IsMax$(score, x)$ then
 9:             $good \leftarrow good + score$
10:         $total \leftarrow total + \max_{y,(x,y) \in asnwers} score(x, y)$
11:     $weights_{unseenX} \leftarrow \sum_{x \notin coveredXs} \left( \max_{(x,y) \in answers} (score(x, y)) \right)$
12:     SetScore$(table, \alpha \cdot \frac{good + table.prior * (weights_{unseenX})}{total + weights_{unseenX}})$
```

APPENDIX

A. UPDATING SCORES WITH EM

Algorithm 2 gives the details of the implementation of the maximization step in Algorithm 1. The score of a table is determined by the ratio of the sum of the weights of correct examples to the count of all examples found in the table.

For each new example generated by the algorithm, we assign a weight equal to the answer score (line 7) computed in the last iteration. Only original examples maintain a score of 1.0 because they are provided by one user. `DataXFormer` exploits the functional nature of the transformation by considering the candidate answer with the maximum score as the correct one (line 8), in order to limit the size of the query results in the next iteration.

We assign a default score for the examples that do not occur in the table to estimate the accuracy of the table on these examples. Finally, the score of each table is multiplied by a smoothening factor α slightly less than 1.0 (line 12) to prevent producing zeroes when calculating answers scores, as explained below. This factor also represents the uncertainty about the rest of the table, resulting from the ambiguity in transformations and table dirtiness.

Algorithm 3 shows the expectation step. We make the simplifying assumption that error rates of the tables are independent, which allows us to calculate the estimated probability that a value y is the transformation of a value x by a simple multiplication. For every value x in X, the probability that y is the transformation of x is computed as the product of the probability of correctness of each table t that supports (x, y), estimated by the score of the table $score(t)$, and the probability of each table $t\prime$ listing another value being wrong, estimated as $1 - score(t\prime)$, where $score(t)$ is the estimated error rate of the table t (lines 4 to 11). We also consider the possibility that all tables are wrong and that none of the provided answers for this x is correct. The es-

Algorithm 3 UpdateAnswerScores

```
Input: $answers$, $tables$
Output: estimated answer probabilities
 1: for all $x \in X$ do
 2:     $A \leftarrow answers.getAnswers(x)$
 3:     $scoreOfUnknown \leftarrow 1$
 4:     for all $table \in answers.getTables(x)$ do
 5:         $scoreOfUnknown \leftarrow scoreOfUnknown \cdot (1 - table.score)$
 6:         for all $(x, y) \in A$ do
 7:             score(x,y) := 1
 8:             if $table$ supports $(x, y)$ then
 9:                 $score(x, y) \leftarrow score(x, y) \cdot table.score$
10:             else
11:                 $score(x, y) \leftarrow score(x, y) \cdot (1 - table.score)$
12:     $sum \leftarrow scoreOfUnknown + \sum_{(x,y)\in A} score(x, y)$
13:     $scoreOfUnknown \leftarrow scoreOfUnknown/sum$
14:     for all $(x, y) \in A$ do
15:         $score(x, y) \leftarrow score(x, y)/sum$
```

timated probabilities are then normalized by dividing them over the sum of the scores of the possible answers as well as the score of the event that none of the answers are correct (lines 12 to 15). The scores are normalized to form a probability distribution, with the highest score being used as an example for the next iteration, with its probability as the weight. Recall that table scores are multiplied by α to avoid zeroes when multiplying probabilities.

B. QUERIES COVERED BY DataXFormer

Table 3 contains in the first column the queries we considered for the experiments in Section 6.2. The subsequent columns denotes whether the transformation was covered by only one table or multiple tables had to be combined, whether the transformation was covered by a Web form, whether the found Web form was wrapped, and the obtained recall, respectively.

Table 3: Queries

Query	Tables found	Form found	Form wrapped	Recall
Fahrenheit to Celsius	no	yes	yes	1.0
miles to km	no	yes	yes	1.0
pound to kg	no	yes	yes	0.8
USD to EUR	no	yes	yes	1.0
zip to state	multiple	yes	yes	0.6
zip to city	multiple	yes	yes	0.34
UPS tracking to address	no	yes	yes	0.2
english to german	no	yes	no	-
swift code to bank	multiple	yes	yes	0.6
hex to RGB	single	yes	no	1.0
ISBN to publisher	multiple	yes	yes	0.8
ISBN to title	multiple	yes	yes	0.8
ISBN to author	multiple	yes	yes	0.8
ISSN to title	multiple	yes	yes	0.8
ip adress to country	no	yes	yes	1.0
domain to primary ip	no	yes	yes	1.0
sentence to language	no	yes	no	-
text to encoding	no	no	no	-
Gregorian to Hijri	no	yes	no	-
CUSIP to company	no	yes	yes	1.0
CUSIP to ticker	multiple	yes	yes	1.0
symbol to company	multiple	yes	yes	1.0
iban to bank name	multiple	yes	yes	0.7
location to temperature	no	yes	yes	1.0
location to humidity	no	yes	yes	1.0
car plate to details	no	yes	no	-
country code to country	single	no	no	1.0
ascii to char	single	yes	no	1.0
car model to brand	multiple	no	no	1.0
country to demonym	single	no	no	0.76
country to language	single	no	no	0.66
country to currency	single	no	no	0.19
company to BBGID	no	yes	no	-
patent ID to name	multiple	yes	no	0.4
city to long/lat	multiple	yes	no	1.0
entity to wikipedia link	no	no	no	-
entity to google graph id	no	no	no	-
person to twitter id	multiple	no	no	0.2
ip to domain	no	yes	yes	1.0
company to CEO	no	no	no	-
company to industry	multiple	no	no	1.0
US standard to metric	single	no	no	1.0
fractions to decimals	no	yes	yes	1.0
country to code	single	no	no	1.0
State to state abbrv	single	no	no	0.95
time zone to abbrv	multiple	no	no	0.9
city to country	multiple	yes	no	0.6
airport code to city	multiple	yes	yes	0.8
RGB to color	single	yes	no	1.0
ASCII to unicode	single	no	no	1.0