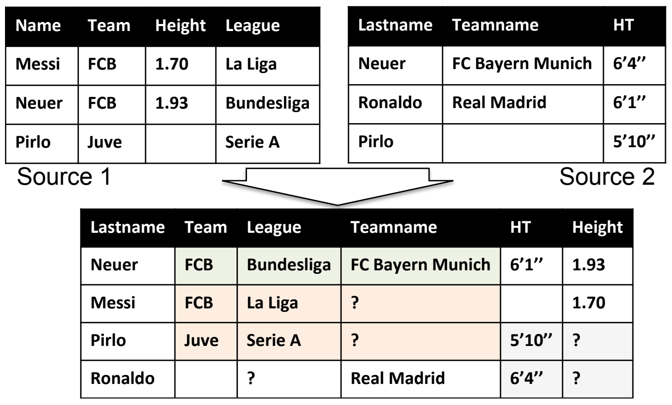
DataXFormer: A Robust Transformation Discovery System

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

***Abstract*—In data integration, data curation, and other data analysis tasks, users spend a considerable amount of time converting data from one representation to another. For example US dates to European dates or airport codes to city names. In a previous vision paper, we presented the initial design of `DataXFormer`, a system that uses web resources to assist in transformation discovery. Specifically, `DataXFormer` discovers possible transformations from web tables and web forms and involves human feedback where appropriate. In this paper, we present the full fledged system along with several extensions. In particular, we present algorithms to find (i) transformations that entail multiple columns of input data, (ii) indirect transformations that are compositions of other transformations, (iii) transformations that are not functions but rather relationships, and (iv) transformations from a knowledge base of public data. We report on experiments with a collection of 120 transformation tasks, and show our enhanced system automatically covers 101 of them by using openly available resources.**


Source 1
Source 2

Name	Team	Height	League
Messi	FCB	1.70	La Liga
Neuer	FCB	1.93	Bundesliga
Pirlo	Juve		Serie A

Lastname	Teamname	HT
Neuer	FC Bayern Munich	6'4"
Ronaldo	Real Madrid	6'1"
Pirlo		5'10"

Lastname	Team	League	Teamname	HT	Height
Neuer	FCB	Bundesliga	FC Bayern Munich	6'1"	1.93
Messi	FCB	La Liga	?		1.70
Pirlo	Juve	Serie A	?	5'10"	?
Ronaldo		?	Real Madrid	6'4"	?

Fig. 1: Values transformation is a critical activity when integrating different sources of data.

I. Introduction

When integrating data from multiple sources there is often a need to perform transformations. These transformations entail converting a data element from one representation to another, *e.g.,* unit, currency, and date format conversions, or generating a semantically different but related value, *e.g., airport code to city name*, or *ISBN to book title*. Consider the simple scenario of Figure 1 where we are integrating two tables containing information about soccer players. We can quickly notice differences in how data is represented in each table. First, one table records player height in meters while the other uses feet and inches. Second, one table stores league and team symbol while the other records team name. Both require transformations. While the first transformation can be computed via a formula, the second requires looking up a dictionary or other data sources.

The Web contains huge amounts of data that can be used as a reference for these transformations. However, it is clearly very tedious for a human to construct such transformations manually, which is one of the reasons why data analysts spend the overwhelming majority of their time "massaging" their data into usable form. Discovering transformations on demand requires a concentrated effort of engineers and domain experts to identify the relevant web sources, understand the relationships among their attributes, and write programs that convey this information in the target table [1]. We aim to support the user by automatically discovering transformations given some input and output examples.

In our initial paper, we showed how to explore web tables and web forms to discover transformations [2]. We used a large corpus of web tables to find the desired user transformations. For static cases, such as converting airport codes to city names, web tables are a useful information source since there are several such conversion tables on the Web. For finding time-varying transformations, such as Euros to Dollars, there are several web forms that will perform them. More recently, we have made our system available on http://www.dataxformer.org and presented a demo at SIGMOD 2015 [3]. Consequently, we have expanded our collection of transformation tasks from 50 to 120. Our initial prototype covered only 52% of the transformation tasks. This is especially due to the limited types of transformations that were supported, focusing only on functional 1-to-1 relationships, while the extended workload contains non-functional and multi-column transformations. Also, our previous approach was limited to the discovery of transformations that appeared in the same table. Thus, we substantially improved `DataXFormer` and were able to increase its coverage to 84%, which is a significant improvement. In this paper, we present the full-fledged system with a focus on algorithms for web tables and the study of the capabilities and the limits of these tables for data transformation discovery. We should note that in practice, our system can be deployed either following the SaaS model on the cloud, where `DataXFormer` runs as a service on the cloud with a RESTful interface, or on-premise, where `DataXFormer` runs on the enterprise computing infrastructure. We begin in Section II with a detailed specification of the transformation problem and the overall architecture of `DataXFormer` with a brief description of previously introduced components. In the following sections, we introduce our new algorithms that are the main contributions of this paper:

- We introduce a general algorithm for discovering tables that contain the desired transformations (Section III). The new algorithm uses an inductive approach to expand the set of web tables that might match the user's request and allows multiple attributes as input values.
- We present an efficient approach to discover suitable join attributes in web tables and hence to identify multi-hop transformations (Section III-C).
- We show how the above algorithms can be adapted to find non-functional mappings, for example the transformation from a soccer team to all of its players, with minimal user verification efforts (Section IV).
- We show how to extend `DataXFormer` to search knowledge bases (KBs) [4] for transformations that are not covered by web tables or web forms (Section V).

Finally, we present a comprehensive evaluation in Section VI where we report on experiments with a collection of 120 transformation tasks, and show that our enhanced system finds 101 of them, with high accuracy. A discussion of related work (Section VII) and future directions of research (Section VIII) conclude the paper.

II. Problem Statement and Overview

In this Section, we give the problem statement for transformation discovery, describe the targeted types of transformations, and present the overall architecture of `DataXFormer`.

A. Problem Statement

The example-driven transformation discovery problem can be defined as follows: Given a set of n example pairs $E = \{(x_1, y_1), \ldots, (x_n, y_n)\}$ that satisfy some hidden relationship R, along with a set of query values Q, we want to find all y_q such that $(x_q \in Q, y_q)$ is a pair drawn from the same relationship R. A pair $(x_i, y_i) \in E$ represents the transformation of x_i into y_i.

Some transformations, such as in feet to meters (see Figure 1), are *syntactic* and can be computed by directly applying a program or a formula to the input values [5], [6]. Other transformations, that we classify to belong to the larger class of *semantic* transformations, cannot be guessed by the input values. They require lookups in some reference data expressing a relationship among the attributes of interest.

Transformations can be many-to-one (*functional*), *e.g.,* zip code to city, or many-to-many (*non-functional*), *e.g.,* books to authors. Figure 2 illustrates different types of transformations based on our running example. Input values can also be composed of multiple attributes as in our example in Figure 2 with x_1="FCB, Bundesliga".

Most existing techniques [4], [7]–[12] that have been proposed to produce a transformation leading to y_q values can be described as a two-stage approach: (1) explicitly model the relationship R between the given x_i's and y_i's, and (2) use R to query the available corpus to produce the instances of y_q. Unfortunately, the first step suffers from serious drawbacks:

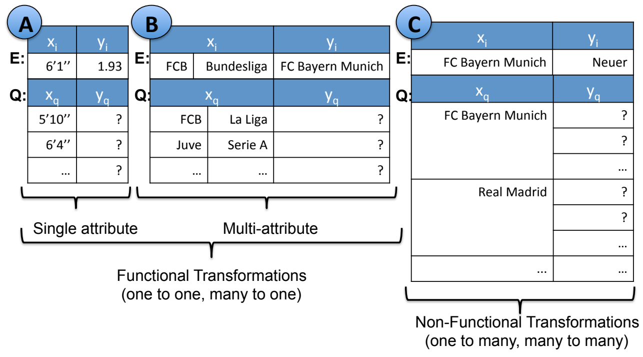


Fig. 2: A: Functional single-attribute transformation: inches to meters. B: Functional multi-attribute transformation: team abbreviation, league to team name. C: Non-functional transformation: team name to team players.

Modeling the explicit relationship R, either as a query [8]–[10], [12] or as a mapping [4], [7], [11], is a hard problem since it requires to discover exact matchings and mappings among the sources [13], [14]. Moreover, these techniques often assume the availability of well curated resources, which is almost never true. In contrast, most useful transformations are usualy scattered among multiple resources, such as web tables and automatically generated knowledge bases that contain errors or do not carry any schema information. Therefore, we propose an example-driven technique for searching and pruning available resources to find the desired transformation results and by doing so, we avoid the task of explicitly modeling the relationship between the available examples.

B. Challenges

Discovering transformations by example on large scale web data poses several unique challenges:

1) Crawling, indexing, and querying the web resources: Every resource type poses specific technical challenges in terms of data collection, storage, and indexing because of the web scale.
2) Complex latent search: Using the examples to locate relevant resources that encode the same hidden relationship. On the one hand, requiring a resource to contain all of the examples at once, e.g., as a conjunctive query, leads to very limited recall, since it is rare that one resource has all the examples. On the other hand, using small subsets leads to resources that encode different relationships due to the ambiguous values that refer to different real-world entities, e.g., "Rome" and "Paris" match both *cities* and *restaurants*, or that express more than one relationship, e.g., "(Messi,La Liga)" matches relationships *debutIn* and *topScorerIn*, but these two relationships return different league values for input value "Ronaldo".
3) Indirect transformation: transformations may not be found directly in any given web resources but may still be uncovered through joins.
4) Ambiguity in non-functional transformations: The number of transformation results might vary for each input value. We have to incorporate an efficient way of supervision to identify false transformations results.

5) Integration of other systems: Plugging external tools as additional providers require to adapt our problem specification to existing query interfaces.

In the next Section, we give an overview of `DataXFormer` and discuss how it tackles the above challenges by using new efficient algorithms with best practice ideas and involving the user or expert into the discovery process.

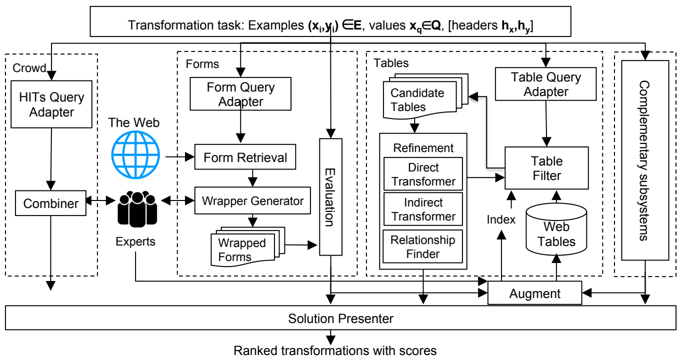


Fig. 3: `DataXFormer` architecture

C. System Overview

Figure 3 illustrates the architecture of `DataXFormer`. Subsystems are deployed for each type of resources with the goal of discovering transformations. Web tables cover the largest set of semantic transformations among the different resources as they explicitly contain the desired transformations and allow to discover multi-attribute and non-functional mappings. Web forms can only be used through the accessible input fields but proved to be very effective at covering single attribute transformations that (i) are based on a specific formula, such as *fraction to decimal*, (ii) are time-dependent, such as *USD to EUR*, or (iii) have an infinite domain size, such as *long/lat to location*. As we will see in Section V, we also exploit KBs to extend the coverage of our system.

`DataXFormer` consumes a transformation task in the form of n example pairs $(x_i, y_i) \in E$ along with a set of m query values $x_q \in Q$, and relevant column headers $h_{X_1}, \ldots, h_{X_k}$ and h_Y, when available. It simultaneously submits this task to each subsystem. In turn, each subsystem first applies a *filter* step that leverages the given examples to identify the most relevant resources, e.g., tables and web sites, and then applies a *refine* step to resolve ambiguities and contradicting results. The user can interact with the system to validate intermediate results or wrap web forms [3]. Once transformation results have been retrieved and scored, successful transformation results, if not already present as a full table, are stored as new tables in the web tables subsystem.

`DataXFormer` also uses expert sourcing for validation and creation of functional transformations [2]. We use the same approach for validating composite and many-to-many relationships. Experts can also be involved in finding transformations for cases where the other resources fail.

Web Tables The web tables subsystem discovers transformations by searching for tables that contain the given examples. The intuition is that these *candidate tables* containing example transformations from E might also contain possible results for the remaining input values $x_q \in Q$. The table retrieval algorithm retrieves candidate tables by using an inverted index that maps each cell value to its table and column identifiers. Candidate results are then analyzed by the refinement component, which verifies the mappings in each candidate table with respect to the given examples. The web tables subsystem discovers functional multi-column, and non-functional transformations, as discussed in Sections III and IV, respectively.

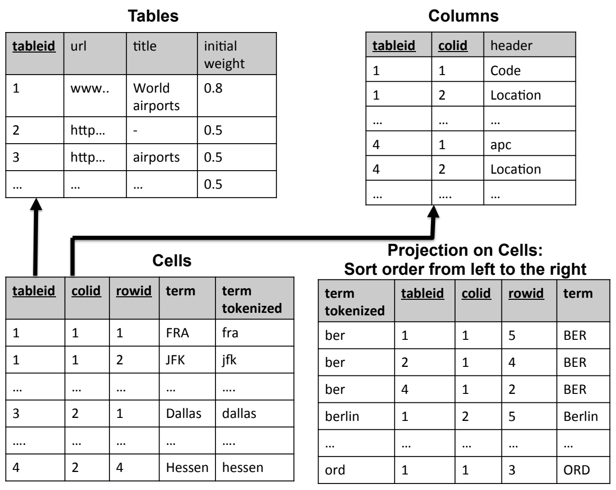
Web Forms We use the same web form subsystem presented in our demonstration [3]. Therefore, we only give a brief overview of the subsystem in this section. A web form is a user interface on a website that allows user-interaction through different fields and buttons. As with web tables, we assume a web form is relevant if it covers some of the n 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 billions of web pages; and (2) any new web form appears to us 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 [15] that both tasks are very hard, even with human intervention. We tackle these challenges by using search engines and a web browser simulator that allows to analyze the HTML code of a form. The user is involved whenever any of the tasks fail. `DataXFormer` maintains a local repository of previously used web forms. The repository is organized as a document store where each document represents one wrapper. If the system fails to find a relevant form in the repository, it uses a search engine to find forms online. 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 the system fails to generate a wrapper or to find the relevant form, the user can be involved [3]. The wrapped forms are tested using the input examples and are evaluated based on their coverage in the *evaluation* step. Candidate web forms are then queried using the input values from Q.

Complementary sources `DataXFormer` can be extended by additional resources, such as knowledge bases, text documents, or code repositories. We have presented ideas on how to involve the crowd in our vision paper [2], and we show in Section V how to use knowledge bases as a complementary resource for transformation discovery.

In this paper, we shed light on the coverage and transformation quality achieved by the subsystems for web forms, web tables, and knowledge bases, and explain their strength and weaknesses for the transformation discovery use case.

III. Single-Column to Multi-Column Transformations

The general workflow of the table subsystem for finding functional transformations is to discover tables that cover a subset of input values in Q. As mentioned in Section II-B, one of the key challenges is to decide when a table is relevant to the transformation task based on the given example pairs in E. The



Tables

<u>tableid</u>	url	title	initial weight
1	www..	World airports	0.8
2	http...	-	0.5
3	http...	airports	0.5
...	...	...	0.5

Columns

<u>tableid</u>	<u>colid</u>	header
1	1	Code
1	2	Location
...	...	...
4	1	apc
4	2	Location
...	....	...

Cells

<u>tableid</u>	<u>colid</u>	<u>rowid</u>	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	<u>tableid</u>	<u>colid</u>	<u>rowid</u>	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

Fig. 4: Schema for storing web tables in a column-store showing an airport code to country example

next challenge is that most tables in the corpus have few rows (on average 12). In a scenario with a small number of examples (x_i, y_i) and a much larger set Q, the system may fail to find the latter by looking only at the relevant tables. Finally, different resources might support different output values for the same $x_q \in Q$. Thus, we need a scoring system that incorporates confidence scores for resources and transformation results.

Given these challenges, our solution is based on an inductive filter and refine approach that leverages intermediate results in multiple iterations. This way `DataXFormer` finds more tables on the fly by extending the set of the given transformation examples with promising intermediate results. In the following, we first discuss the index structure and the filtering mechanism and how they can be easily generalized for multi-column input values. Then we discuss the inductive approach and the accompanying refine process to consolidate transformation results.

A. *Indexing Web tables*

Traditional keyword search applications rely on predefined foreign key relationships and specialized indexes that require domain knowledge and context information. Since web tables are heterogeneous, differ in schema and size, and oftentimes lack column labels, we store them in the most general way. Ideally, we require to have an index on all possible input values to find relevant tables. In our previous paper, we compared the storage of web tables as documents to a universal main table in a column store [2]. The later storage scheme, described below, turned out to be the most efficient option.

Relation $Cells$ in Figure 4 represents every cell of a web table by a tuple with the IDs for table, column and row, along with the tokenized, stemmed form of its value. We also maintain additional metadata for each table, such as column headers, source URL, title, and an initial weight expressing its authoritativeness. The initial weight influences the confidence score we compute to rate transformation results.

We store the web tables in a multi-node Vertica instance. Vertica employs projections (sorted views) on tables in place of indexes. While the table itself is a projection that is stored as depicted in Figure 4, we use another projection on relation $Cells$ sorted on the tokenized values. The two projections allow us to swiftly retrieve relevant table IDs for tokenized input examples, and to easily load the content of a table based on its table and column IDs. An additional trie index allows fuzzy matching when the given transformation examples and the values in the database differ with a small edit-distance.

B. *Querying Web Tables*

While the index on input values enables the system to retrieve all tables that contain a specific value, it is important to filter irrelevant tables as early as possible. This is crucial as we are working on millions of tables, which means that we want to retrieve only those tables that contain a relevant subset of examples in E. Our original prototype was optimized to identify tables with two columns that contain at least τ of the given example pairs $(x_i, y_i) \in E$. In our previous experimental study on 50 transformation tasks, $\tau = 2$ proved to be a reasonable filter for avoiding irrelevant tables and meeting a sweet spot in terms of precision/recall [2].

```
SELECT colX1.tableid, colX2.colid, [colX2.colid,...], colYid
FROM
            (SELECT tableid, colid
             FROM Cells
             WHERE term_tokenized IN (X1)
            GROUP BY tableid, colid
            HAVING COUNT(DISTINCT tokenized) >= tau) AS colX1,

            [(SELECT tableid, colid
            FROM Cells
            WHERE term_tokenized IN (X2)
            GROUP BY tableid, colid
            HAVING COUNT(DISTINCT tokenized) >= tau) AS colX2, ...]
            (SELECT tableid, colid
            FROM Cells
            WHERE term_tokenized IN (Y)
            GROUP BY tableid, colid
            HAVING COUNT(DISTINCT tokenized) >= tau) AS colY

WHERE colX1.tableid = colY.tableid
            [AND colX1.tableid = colX2.tableid AND ...]
            AND colX1.colid <> colY.colid
            [AND colX1.colid <> colX2.colid AND colX2.colid <> colY.colid
            AND ...]
```

Fig. 5: Black: code for finding single column input values. Blue: Additional code needed for composite input values.

The universal main table structure enables us to submit only one query for the retrieval of all tables that contain at least τ of the input/output pairs. The query in Figure 5 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. The result of the query is a set of triples, each comprising a table ID and two column IDs. To maximize coverage, we tokenize and stem every value x and y from the input.

Using the above query skeleton, adapting the table retrieval component for multi-attribute input values $x_i =$

$X1, X2, \ldots, Xk$ is straightforward. We simply extend the query with more subqueries; one for each additional input column. The extensions are outlined by means of square brackets and blue color. For every additional input column, the query has to generate a subquery and additional checks in the `WHERE` clause. The checks in the `where` clause enforce that the discovered columns in the web tables are different from each other. While identifying candidate tables this way is very fast, we still need to make sure that the identified tables contain the example values in the appropriate rows.

The cost of processing the above query increases with the number of examples. This issue becomes pertinent as we follow an inductive approach. After a *filter-refine* iteration, we use the retrieved intermediate results to find additional tables that eventually cover the remaining $x_q \in Q$. Since not every intermediate result is correct, they do not necessarily contribute to the recall of our system. Additionally, querying for a large number of examples hurts the performance without a significant gain. We therefore limit the set of intermediate results to the top k based on confidence scores that we compute and update after each filter-refine iteration.

C. Joining tables

In the previous subsection, we only addressed the discovery of tables that maintain both the input and the output values of a transformation example. We refer to the strategy based on this assumption as *direct* transformations. However, the actual transformation input and output columns could reside in different tables and be linked through other columns not provided in the transformation task. This can happen if the tables have a foreign key relationship or an inclusion dependency through two columns that share some values Z. We deem transformations that were discovered based on joined tables as *indirect transformations*. Unfortunately, we do not have complete knowledge of the underlying schema, therefore we cannot rely on existing algorithms, such as [16]–[18], that rely on specialized indices, predefined foreign key-primary key relationships, or meta-data information and dependencies.

To find the transformation result in this scenario, it is necessary to join at least two tables and align the input and output values accordingly. However, a brute-force search of all possible inclusion dependencies among all web tables that contain the actual input values X would be prohibitive. This problem is even more striking when more than one table has to be joined to link an example input value x to its output value y. Accordingly multiple intermediate inclusion dependencies lists $Z_1, Z_2, \ldots, Z_k$ have to be discovered to serve as join attributes. Our intuition to reduce the search space, without missing useful joins for the transformation at hand, is to require that the values z_q (i) are functionally dependent on the input values x_q, and (ii) functionally determine the transformation results y_q, simultaneously.

Algorithm 1 searches for joinable tables in a breadth-first manner. The algorithm consumes as input the initial examples E and the values $x_q \in Q$. With each iteration, the indirection path increases until the maximum length of an indirection path has been reached, or all x_q values have been covered. The algorithm starts ($path = 1$) by identifying all tables that contain the x values of the provided examples (line 4). After excluding the tables that provide a direct transformation of $(x_i, y_i) \in E$, the algorithm checks each remaining table in T_X for potential join columns Z_i (line 7). A column can serve as a join column if a functional dependency $X \rightarrow Z_i$ holds with regard to the input values $E.X$ that are covered by the table at hand. This check is performed in *findJoinColumns*, which requires a pairwise FD validation test with regard to the values in other columns. If an FD between the column that contains our input values X and another column Z in the current table T holds, we extract the subset of values that appear in the same rows as our given input values X and call it Z_i. Furthermore, we check whether the cardinality of the discovered Z_i is greater or equal than the corresponding $E.Y$. If this is not the case, we know apriori that the functional path to the transformation result cannot be maintained.

Algorithm 1 tableJoiner

```
Input: Initial examples $E$, a set of input values $Q$, initial tables $T_E$
Output: tables
 1: $path \leftarrow 1$
 2: repeat
 3:   if $path = 1$ then
 4:     $T_X \leftarrow$ QueryForTables($E.X$)
 5:     $T_X \leftarrow T_X \setminus T_E$
 6:     for all $T \in T_X$ do
 7:       $[Z_0, Z_2, \ldots, Z_n] \leftarrow$ findJoinColumns(T, E.X)
 8:       for all $Z_i \in [Z_0, Z_2, \ldots, Z_n]$ do
 9:         $T_J \leftarrow$ findJoinableTables($Z_i, E, T$)
10:         if $T_J \neq \emptyset$ then
11:           if $T_j$ covers $\tau(x_i, y_i) \in E$ then
12:             $tables$.Append($T_j$)
13:         $currentTablePaths$.Append($path < T, Z_i >$)
14:   else
15:     $newTablePaths \leftarrow \emptyset$
16:     for all $< T_p, Z > \in currentTablePaths$ do
17:       $T_Z \leftarrow$ QueryForTables($Z$) $\setminus$ previously seen tables
18:       for all $T_z \in T_Z$ do
19:         $T_z \leftarrow T_z \bowtie_Z T_p$
20:         $[Z_0, Z_2, \ldots, Z_n] \leftarrow$ findJoinColumns($T_z, Z$)
21:         for all $Z_i \in [Z_0, Z_2, \ldots, Z_n]$ do
22:           $T_j \leftarrow$ findJoinableTables($Z_i, E, T_z$)
23:           if $T_j \neq \emptyset$ then
24:             if $T_j$ covers $\tau(x_i, y_i) \in E$ then
25:               $tables$.Append($T_j$)
26:             $newTablePaths$.Append($path < T_j, Z_i >$)
27:     $currentTablePaths \leftarrow newTablePaths$
28:     ClearFrequentedTables($currentTablePaths$)
29:   $path = path + 1$
30:   $alltablePaths$.Append($currentTablePaths$)
31: until $|path| > maxLength \vee X = scoredAnswers.X$
```

Next the algorithm searches for tables that can be joined to the current table T based on each of the discovered IND attributes Z_i. If the table also contains correct mappings to Y, we store the table as a new joined resource table. $findJoinableTables$ creates and filters joined tables that maintain the approximate mapping $E.X \rightarrow Z \rightarrow E.Y$ for τ of the examples in E . Next, all joined tables in T_j are checked for indirect transformations. We store the table that contains the mapping $X \rightarrow Z$ if longer paths need to be processed.

In later iterations ($path > 0$), the procedure is repeated by starting the search from the most recent set of Z values in joined tables T_j that were discovered in the previous path.

We treat each set Z as our current set of input values. This maintains the semantics of the transformation as each consecutive join maintains the functional dependency. As we are following a breadth-first traversal of the joinable tables graph, the set of Z values increases exponentially after each iteration. Our strategy to reduce the search space is to require the functional dependency constraint on at least τ of the initial examples throughout the indirection and no contradiction with regard to the given examples, as shown in Algorithm 1 (line 6). The breadth-first approach ensures that `DataXFormer` finds transformations from shorter indirections first, which is desirable considering that shorter indirection paths might constitute stronger relations than longer indirection paths.

D. Scoring and Ranking Transformations

In the *refine* phase, we load the content of each candidate column pair and check whether the values from a pair (x_i, y_i) in the input query appear in the same row in the candidate table. If τ examples still match after considering the row correspondence of the discovered (x_i, y_i) pairs, `DataXFormer` collects all transformations y_q for x_q's that reside in the same table.

The retrieved tables may provide conflicting answers by returning different y_q values for the same x_q input. A naive reconciliation solution would be to deliver the most frequent y_q for each given x_q (majority voting). However, this approach suffers from its inability to properly score results and tables while taking into account the iterative nature of the process. Instead, we propose a solution with the following desiderata: First, it is necessary to compute confidences scores for the tables to estimate their authoritativeness and rate their success in covering examples. Second, as we incorporate results of previous iterations as additional examples, we also need a confidence score for those examples. Third, in each iteration we need to recompute the scores of previously found transformations. Finally, it should be possible to incorporate additional score components based on the given dataset, such as schema similarity, provenance-based scores, or user feedback.

We adopt an iterative expectation-maximization (EM) approach [19] that incorporates confidence scores. The confidence score of each table, i.e., the probability it provides a correct answer, is estimated based on the current belief about the probability of the answers. Initially each table is assigned with a score based on the number of user examples it covers and the Jaccard similarity of labels, if available. The score of the table is weighted with an initial weight, which is stored in relation $Tables$, and either has a default value or a value assigned by an expert. The answer scores are updated based on the newly computed table 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 value $x_q \in Q$, the scores of possible answers form a probability distribution. The described process works for both types of functional transformations, *i.e.,* multi-attribute and single attribute. All components of a multi-attribute input value have to appear simultaneously to be considered.

Algorithm 2 describes the reconciliation process based on EM and embedded within the overall workflow of the system's filter-refine iterations. The maximization and expectation steps are described in Algorithms 3 and 4, respectively.

Algorithm 2 Expectation-Maximization

```
Input: A set of initial examples $E = \{(x, y), \ldots\}$, input values $Q$
Output: Scored answers
1: $answers \leftarrow E$
2: $tables \leftarrow E$
3: $finishedQuerying \leftarrow false$
4: $oldA \leftarrow null$
5: repeat
6:     if not finishedQuerying then
7:         $tables \leftarrow$ QueryForTables($answers$)
8:         $tables \leftarrow tables \cup tableJoiner(answers, Q, tables)$
9:         for all $t \in tables$ do
10:            for all $answer(x_q, y_q) \in t$ do
11:                if $x_q \in Q$ then
12:                    UpdateLineage($t, answer$)
13:                    $answers$.Add($x_q, y_q$)
14:        if not new $x_q$ was covered by tables then
15:            $finishedQuerying \leftarrow true$
16:    UpdateTableScores($answers, tables$) //maximization step
17:    UpdateAnswerScores($answers, tables$) //Expectation step
18:    $\Delta scores = \sum |answers.score(x, y) - oldA.score(x, y)|$
19:    oldA = answers;
20: until $finishedQuerying \wedge \Delta scores < \epsilon$
```

In each iteration of Algorithm 2, we query for tables using the new weighted examples (line 7), until no more values in Q can be covered (lines 6 and 15). We also incorporate in line 8, tables that we joined on the fly. We treat each joined table as a table on its own. In line 16, we implement the maximization step by updating table scores (estimated error-rates) based on the current belief in the answers (answer scores). At the bootstrap stage, initial scores are assigned based on the percentage of user-given examples that were covered by a table. In each iteration, if new unseen tables are 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. The system converges when the total (absolute) change in answer scores is below a small value ϵ (line 20).

The score of a table is determined by the ratio of the sum of the scores of correct examples to the sum of all examples that are believed to be correct. For examples in the table, we retrieve the answer score (line 8 in Algorithm 3) computed in the previous maximization step. Original examples maintain a score of 1.0 since they are provided by the user. The test $IsMax$ checks whether the score of the current pair (x, y) is the pair that has the highest score among all pairs with input value x. If this is the case, we assume that according to the current belief (x, y) is a correct mapping and we increment the counter $good$ with its relative score. Otherwise, we increment bad by 1. We assign a default score for the example input values that do not occur in the table, $unseenX$ to estimate the relevance of the table on these examples. All scores are weighed with the table prior $prior_t$, that is either set by a user or set to a default value of 0.5.

Finally, the score of each table is multiplied by a smoothening factor α slightly less than 1.0 (line 14) to prevent producing zeroes when calculating answers scores. This factor also

Algorithm 3 UpdateTableScores

```
Input: $answers$, $tables$
Output: estimated table scores
1:  for all $t \in tables$ do
2:      $good \leftarrow 0$
3:      $bad \leftarrow 0$
4:      $total \leftarrow 0$
5:      $coveredXs \leftarrow \{\}$ //holds example x's appearing in t
6:      for all $answer(x, y) \in t$ do
7:          $coveredXs \leftarrow coveredXs \cup \{x\}$
8:          $score \leftarrow$GetScore$(x, y)$
9:          if IsMax$(score, x)$ then
10:             $good \leftarrow good + score$
11:         else
12:             $bad \leftarrow bad + 1$
13:     $unseenX \leftarrow \sum_{x \notin coveredXs \wedge (x,y) \in answers} \max\big(score(x, y)\big)$
14:     SetScore$(t, \alpha \cdot \frac{prior_t \cdot good}{prior_t \cdot good + (1 - prior_t) \cdot (bad + unseenX)})$
```

Algorithm 4 UpdateAnswerScores

```
Input: $answers$, $tables$
Output: estimated answer probabilities
1:  for all $x \in X$ do
2:      $A \leftarrow answers.getAnswers(x)$
3:      $scoreOfNone \leftarrow 1$
4:      for all $table \in answers.getTables(x)$ do
5:          $scoreOfNone \leftarrow scoreOfNone \cdot (1 - table.score)$
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:     for all $(x, y) \in A$ do
13:         $score(x, y) \leftarrow \frac{score(x,y)}{scoreOfNone + \sum_{(x,y) \in A} score(x,y)}$
```

represents the uncertainty about the rest of the table, resulting from the ambiguity in transformations and table dirtiness.

Algorithm 4 shows the expectation step. We make the simplifying assumption that the error rates of the tables are independent. Such assumption 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 $x \in Q$, 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 none of the provided answers for this x is correct by aggregating *scoreOfNone*. The estimated 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 13). 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.

IV. From n:1 Transformations to n:m Relationships

In functional transformations, the fact that there is only one correct answer was used to prune many resources based on the given set of examples E and to keep only the most likely transformation value y_q for an x_q. However, in many-to-many relationships, a newly found y'_q can be another correct mapping. In order to cover many-to-many (i.e., non-functional) transformations, we have to adapt our scoring system to allow multiple solutions for one input value, e.g., *countries* to k *cities*. This is particularly hard for two main reasons: We do not know how many values are correct for a given x_q. Restricting the output to a fixed number of values limits the recall of the transformation. On the other hand, accepting all possible mappings will result in extracting relationships that do not adhere to the actual transformation task. This easily leads to mixing heterogeneous values in the result. This of course impacts the precision of the transformation. We should note that while it is possible to discover more functional transformations by joining tables on the FD constraint (Section III-C), applying indirect transformations for non-functional transformations would require a completely different strategy as the pruning strategy via FDs cannot be adopted. We leave this for future work.

In the following, we discuss two approaches for the general $n{:}m$ mapping problem. The first automatically clusters mappings into correct or false ones based on their score distribution. The second, implemented in `DataXFormer`, asks the user for feedback on effectively chosen sample results.

A. Automated Clustering

Once the candidate y_q values for a x_q are retrieved and scored, one way to identify the set of interest is to use a threshold. However, we would have to chose an arbitrary threshold that only holds for some use cases. Another way is to use clustering to separate y_q values with high scores and y_q values with low scores. We start by defining the distance between two scores as their absolute difference.

Definition 1: Let y_1 and y_2 be two candidate values for an input value x_q and c a function to get their score, the distance between x_1 and x_2 is: $D(y_1, y_2) = |c(y_1) - c(y_2)|$

Two values with high scores are expected to have a smaller distance between them than the distance between a high-scoring value and a low-scoring one. Our goal is to obtain an optimal separation between high- and low-scoring values.

Let H be the set of high-scoring values and L the set of low-scoring ones. Intuitively, a separation is preferable to another one if by adding a value $x \in L$ to H, the difference between the sum of pair-wise distances among all values of $H \cup \{x\}$ and the sum of their scores becomes smaller. The intuition is clarified in the following gain function.

Definition 2: Let S consist of the candidate values for $y_i \in Y_q$ and L be a subset of S. We define the *gain* of L as the sum of all scores of elements in L subtracted by the sum of all score distances among elements in L:

$$G(L) = \sum_{s \in L} c(s) - \sum_{1 \le j < |L|} \sum_{j < k \le |L|} D(y_j, y_k)$$

We define an optimal separation as the one that maximizes this function for H. As an optimal separation requires an exponential exploration over the possible subsets, we use a nearest neighbor chain algorithm for agglomerative clustering [20]. We first order tuples in S in descending order wrt their scores, and create a cluster for each value. We pick the highest scoring cluster, and keep adding to it the next value in the order, while computing the separation gain at each step. We terminate after reaching a separation where the gain attains a local maximum. The algorithm requires a linear space and quadratic time (pair-wise distances) in the number of tuples.

B. Generating Validation Samples for n:m Relationships

The major flaw in the above approach is that it does not have the opportunity of taking into account any negative examples that would penalize irrelevant resources and their candidates. In the absence of any known negative information, we need to ask the user for feedback. Thus, we extend our web table system for discovering functional transformations with a feedback round to the loop. After every iteration, some mappings are selected for validation, and the user's feedback is propagated to prune tables with incorrect mappings. For this purpose, we slightly modify the formula in line 14 of Algorithm 3 to:

$$\alpha \cdot \frac{prior_t \cdot good - 2 \cdot (1 - prior_t) \cdot bad}{prior_t \cdot good + (1 - prior_t) \cdot (bad + unseenX)}$$

The variable bad refers to the number of false mappings contained in a table. To be conservative, this number is doubled. Whenever the score is negative, we drop the table completely as a supporting source. The user feedback after each filter and refine iteration also prevents the system from using wrong transformation results as examples for the next iteration. In our experiments, the algorithms usually converged before the 5^{th} iteration, when asking the user for 10 validations per iteration.

The challenge is then how to identify the most useful set of examples for user validation. Similar to any other classification task, validating a transformation sample can improve the precision of our approach only if the sample represents the whole set of discovered transformations. For our approach, whenever false mappings are retrieved in the filter phase, it is desirable that some of them appear in the validation sample. This way, it is possible to identify tables and sources that lead to false mappings, which are more critical for the quality of the approach than the confirmation of correct mappings. We feed these gained insights into the EM model and recompute the scores of tables and answers. In the following, we compare different policies for selecting the mappings to validate, with the goal of maximizing the number of false mappings.

Frequency-based Selection A mapping between two values can appear in multiple tables. If a mapping has a high frequency, it co-occurs in several tables given the user-given examples. The frequency could be considered as a score for the relevance of identified mappings. Hence, a natural intuition to identify false mappings for the validation is to expose low-frequent ones to the user. Of course, this approach might overlook prominent but irrelevant mappings. Penalizing a rare mappings does not have a strong effect on the learning process because it affects only few tables.

Score-based Selection Following the intuition of relevance in the frequency-based selection method, one can also incorporate the actual answer scores that have been computed by the expectation maximization model in the functional scenario from Section III-D. Notice that the expectation maximization model scores tables and their mappings based on the ratio of existing true and false mappings. Although, in the $n{:}m$ scenario, false mappings are not known before the first user validation, we can still rank the answers by scores because the coverage of true values affects the score.

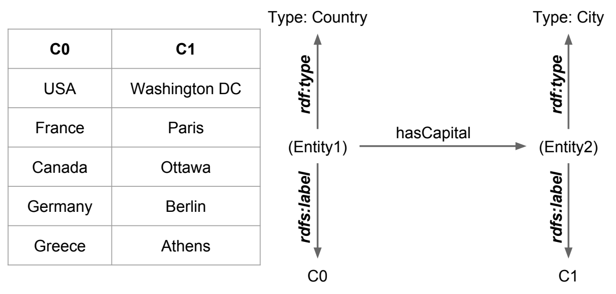
Diversified Selection A further intuition to generate a validation set is to represent as many tables as possible. Thus, example selection can be modeled as a maximum coverage problem, where each example is associated with the set of tables that contain it: given a number of examples (sets) we want to maximize the number of covered tables (the elements). This problem is NP-hard, but the simple greedy algorithm has linear complexity and guaranteed approximation ratio. However, the maximum coverage strategy is likely to lead to a validation sample of very frequent mappings that also co-occur in many tables. This way it is not possible to distinguish tables from each other. Thus, we prioritize mappings that have a minimum overlap *w.r.t.* their containing tables. We extend the problem to the *weighted* version, where we want to maximize the number of covered tables and the number of non-overlapping tables (the weight). For this purpose, we start with the least frequent mapping and greedily add mappings that have the least overlap in tables with the already chosen mappings. The weighted version of the algorithm maintains the same approximation ratio of the standard version. In Section VI-C2, we compare the above proposals and show that the diversified selection based on minimal overlap yields the best results.

V. Complementing Approaches

`DataXFormer` can also be complemented with other querying systems. A compelling example are alternative systems that try to discover relationships by mapping relational tables to knowledge bases. In the following, we outline how we adapted a data curation system that relies on knowledge bases for our transformation discovery problem.

Knowledge Bases for Transformations. Human-curated knowledge bases (KBs) contain high-quality information, which, intuitively, can lead to high precision in the discovered y_q values even in the presence of one source only. In fact, differently from the web tables, it is possible to apply discovery algorithm to *explicitly* identify the underlying relationship in terms of the target ontology [4]. An obvious drawback of knowledge bases is that they usually only cover prominent head-topic transformations, thus limiting the general coverage. Moreover, a knowledge base usually suffers from inconsistencies that affect the accuracy of discovered patterns [21].

We leverage recent developments in the discovery of mappings between a relation and a KB [4]. This system generates candidate mappings to express the underlying relationship in terms of the knowledge base, based on support and internal coherence. A mapping is a graph whose edges are properties



C0	C1
USA	Washington DC
France	Paris
Canada	Ottawa
Germany	Berlin
Greece	Athens

Fig. 6: Mapping an input query to a knowledge base

(relationships) in the knowledge base and its internal nodes are types or placeholders for entities in the knowledge base. Its end nodes are literals that corresponds to attributes in the example given as input. A mapping applies to a given example if (i) there are entities in the KB that align with the connected types and relationships, and (ii) the corresponding literals at the end nodes match the values in the example. For example, consider the relation in Figure 6 reporting a list of countries with their capitals. On the right hand side of the figure, the reported mapping holds for the first tuple if there is a triple ("USA",*hasCapital*,"Washington DC") in the KB.

Two-stage Discovery. Given a trusted KB, in the first step, we use the given examples to discover the explicit mapping between the dataset and the KB. As for the matching of the tables, we are interested in mappings that hold for a minimum number of examples. It is possible that multiple patterns apply to the example mappings. In case they have the same support, we favour more restrictive patterns. Consider again Figure 6. Assume we have a second relationship *hasCity* that matches our examples. *hasCity* clearly subsumes *hasCapital*. If they have the same support (i.e., they are all capitals), we would prefer the first, which is more restrictive (i.e., has a lower frequency). In our study we take the top-1 mapping for functional transformations and use the automated clustering approach in Section IV for non-functional mappings.

In the second step, the discovered relationship is used for each x_q, e.g., "France", to find all the y_q values related to it through this mapping in the KB, e.g., "Athens".

Implementation. For fast navigation and querying of the KB, triples in `DataXFormer` are indexed by all components: subject, object and predicate. Denoting the subject by 'S', the predicate by 'P' and the object by 'O', the resulting six indices are SPO, SOP, PSO, POS, OSP, and OPS. Literals always fall into objects of triples. The main disadvantage of this setting is that storing a KB takes 6 times its original size, but it allows fast querying by any part of a triple. Our implementation works on top of YAGO [22] and DBpedia [23], two general purpose KBs.

Mapping to curated resources, such as a KB, can return high quality transformations if the mapping is encoded in the resource. The problem, however, is coverage, as a general purpose KB usually only covers head topics, as we show next.

VI. Case Study

To evaluate our system, we collected 120 transformation tasks from real world users and experts from a data curation company, more than doubling the size of our previous study [2]. In the following, we describe our experimental setup, discuss the coverage of the Web resources, and finally report the quality of the transformations at the instance level.

A. *Experimental Setup*

Our evaluation refers to 120 transformation tasks that comprise 79 functional single column, 10 multi-column, and 31 non-functional transformation tasks. We manually collected ground truth for each of the tasks with a finite domain, such as *city and zipcode to country*, and generated random input values for formula based transformations, such as *fraction to decimal*.

We use the Dresden Web Table Corpus [24], which contains about 120 millions web tables from about 4 billions web pages. Additionally, we cleared the tables from all text and comment columns with more than 200 characters to reduce storage and runtime. The tables are stored in a four node Vertica DBMS consuming 700 GB of disk space including all projections. The knowledge base subsystem, adapted from Katara [4], is tested on top of the most recent versions of the YAGO and DBpedia KBs.

We consider two metrics: (i) coverage and (ii) quality. The coverage refers to the fraction of transformations for which a system was able to find relevant web tables, web forms, and mappings with a KB. The quality assesses the correctness and completeness of a transformation task in terms of precision and recall.

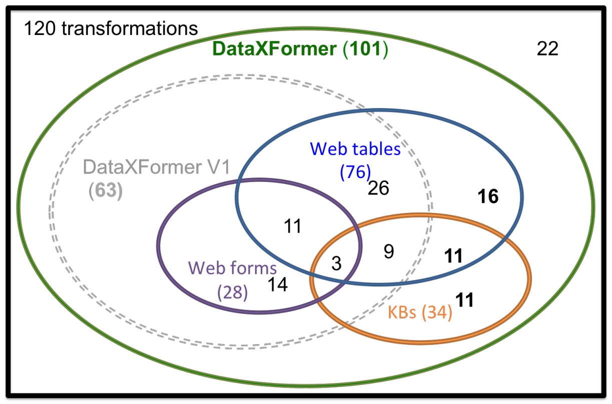


Fig. 7: The Venn diagram illustrates the number of transformations covered by each resource

B. *Coverage of Web Resources*

Figure 7 depicts a diagram on how many of the 120 transformation tasks could be covered by the given resources. The number of covered transformation tasks are put between parentheses in front of each resource while the other numbers show the overlaps. A transformation is deemed to be covered if our system finds a resource, such as a web table, that can at least partially solve the task, *i.e.,* there is a table that contains at least three pairs of input/output values from the ground truth.

As we can see, most of the transformation tasks (76) can be covered by web tables, followed by KBs and web forms.

Interestingly, the web tables nearly completely subsume the coverage of the KB subsystem. Only 11 transformations could be exclusively covered with KBs, including transformations, such as *soccer player to birth place*, *soccer player to birthdate*, or *country to head of state*. This is expected since discovering exact, complex relationships is hard without human supervision, and the KBs have strong evidence only for head topic transformations. The web form subsystem covers 28 of the transformation tasks, all of which are functional single column transformations, including for example *celsius to fahrenheit*, *airport code to city*, or *ip address to domain*. Among those transformation tasks, 14 could only be covered by web forms, mostly relating to formula based transformations.

Figure 7 also illustrates the difference in coverage between DataXFormer and its previous version. The new system covers 35 more transformations. In particular, 28 transformations are covered by means of the newly proposed approaches, namely, indirect discovery, multi-column, and non-functional transformations, and 11 with the addition of a KB system. Transformations that could not been covered by any of our web resources, included *English to German*, *sentence to language*, *text to encoding*, *Gregorian to Hijri*, *car plate to details*, *company to Bloomberg id*, *bank to country*, *bank+city to swift*, *bank to city*,*bank to swift code*, for which we could find a web form but could not wrap the web form with our system.

TABLE I: Coverage by transformation type

Approach	#	tables	forms	KB	Joined coverage
Single Column	79	50 (63%)	28 (23%)	17 (22%)	68 (86%)
Multi Column	10	7 (70%)	0	0	7 (70%)
Non-functional	31	19 (61%)	0	17 (55%)	25 (81%)
Total	120	76 (63%)	28 (23%)	34 (28%)	101 (84%)

Table I shows the coverage with regard to the transformation task type. While multi-column tasks could only be covered by the new web tables subsystem, non-functional transformations could be covered by both the tables and the KB subsystems. Compared to the its original version, DataXFormer now also covers one more functional transformation by means of its indirect transformation algorithm. The transformation *Unesco site to country* did not appear in any web table explicitly, but could be covered by joining two tables, each containing one side of the transformation, on columns with *country ISO codes*. We report the indirection result with max path length of 1. Our studies showed that allowing a path length higher than 1 did not improve the results. At the same time, the runtime was strikingly high. While the algorithm runs a couple of minutes for path length 1, it takes several hours for path length 2.

C. Transformation Quality

We measure the quality of each transformation result *w.r.t.* precision and recall. In our previous study, we reported experiments on the effects of the threshold τ and of the number of examples n [2]. The main insight is that $\tau = 2$ leads to the highest harmonic mean (F-Measure) of precision and recall on our workload. Threshold $\tau = 1$ leads to more false positives, resulting in low precision, while higher values for τ penalizes the recall. We found that the average precision is above 90% for any number of examples n, while the recall increases steadily with higher values, e.g., 32% for $n = 2$, 70% for $n = 5$, and 80% for 20 to 40 examples.

In this paper, we discuss the performance of each subsystem on functional transformations. We then show the impact of different example selection strategies for semi-automatic discovery of non-functional transformations. For all the following experiments, we set $\tau = 2$ and $n = 5$.

TABLE II: Average precision and recall of each subsystem

Subsystem	Precision	Recall	#
Web tables	0.80	0.76	57
Web forms	0.96	0.83	28
KB	0.32	0.19	17
best	0.87	0.76	

1) Functional Transformations: Table II illustrates the average precision and recall values for all functional transformations that were partially covered by the respective subsystem. For all the experiments, we used 5 randomly chosen transformation examples from the available ground truth. Surprisingly, the KB subsystem shows worse performance than the web tables and web forms subsystems. There are several reasons for this result. First of all, KBs can suffer from incompleteness and inconsistencies in the use of the relationships, e.g., the properties *location* and *locationCity* are used interchangeably for the same semantics [21]. In addition, the same property might have values with completely different semantics, e.g., *location* might point to a city, a region, or even a country. This ambiguity may lower the recall, as the property path would not always lead to the desired transformation.

Aggregating the precision and recall of the best component for a specific transformation results in 87% precision and 76% recall. A web form usually yields high precision results as the corresponding transformation tasks implement a formula, which always returns the correct transformation upon a successful wrapping. Web tables yield very high results on tasks where the complete domain of the transformation fits in few overlapping tables. For example, there are single tables that contain transformations, such as *university to state* (0.93 Pr/0.93 Re), or *airport code to city* (1.0 Pr/0.80 Re).

TABLE III: Majority voting vs. EM propagation system

Method	Precision	Recall	F-Measure
Majority voting	0.56	0.68	0.66
EM model	**0.80**	**0.76**	**0.78**

Table III compares the EM scoring model against a majority voting approach that scores answers by their frequency. The table denotes the average precision and recall over all functional transformations and shows that EM model significantly improves the performance over a naive voting approach.

TABLE IV: Average precision and recall of web tables algorithms for functional transformations

Algorithm	Precision	Recall	#
Direct single column	0.83	0.76	49
Direct multi column	0.95	0.67	7
Indirect single column	0.79	0.56	14

Table IV additionally breaks down the results based on the approach that is applied to solve the task. For this experiment,

TABLE V: Average precision and recall of different approaches for the non-functional transformations

Approach	Precision	Recall	FMeasure
supervised			
Frequency based (least)	0.82	0.57	0.67
Frequency based (most)	0.69	0.62	0.66
Score based	0.72	0.62	0.67
Diversified (coverage)	0.84	0.56	0.67
Diversified (least overlap)	**0.86**	0.58	**0.69**
unsupervised			
Automated clustering	0.45	0.25	0.3
Bayesian network	0.35	**0.63**	0.45
KB	0.3	0.25	0.27

we run our system twice for each covered single column transformation task: first considering only direct transformations and then considering only joined tables. As expected, the indirect transformation approach yields poorer precision and recall results if performed in isolation. Therefore, the best way for finding useful transformations is to try to find direct mappings first, and then to turn to indirect transformations for covering values from Q that have not been covered.

2) Non-Functional Mappings: We study the impact of 3 supervised and 3 unsupervised methods for the non-functional transformations. In particular, we use the unsupervised clustering and the supervised selection strategies over the web corpus from Section IV-B, the unsupervised KB subsystem, and a state of the art approach based on Bayesian networks for discovering set of values from web resources [25]. In particular, the Bayesian approach learns true positive and false negative rates of sources by sampling from skewed Beta and Bernoulli distributions, but does not assume any ground truth as input. Therefore, we adapted it to incorporate the given examples as true facts without sampling their probability.

Table V reports averaged results for every method on the subset of 31 non-functional transformations that were covered by that subsystem. Each transformation was executed with 3 supervised iterations. In each iteration, the user validated 10 example results. The strategy for diversified example selection based on minimal overlap yields the best results. The bad recall performance of the clustering approach is caused by the exclusion of supposedly wrong results (false negatives) as it does not allow rare correct transformations to be captured in the results. Despite the examples, the inherent assumption of global truth of prior distributions in the Bayesian approach leads to arbitrary samples of truth probabilities that overestimate the rating of a table leading to low precision but relatively high recall. The approaches on top of web tables outperform the KB approach in terms of precision, because ambiguities and inconsistencies in KBs are present also in the non-functional scenario.

Figure 8 shows why the approach based on minimal overlap yields the best results. It selects on average more incorrect examples, which can be flagged as wrong transformations by the user and ultimately lead the algorithm to assign correct scores to the sources. At the same time, it affects more tables with its maximization effort than the approach that selects the least frequent or least scored transformation results.

Figure 9 shows the effect of the number of iterations on the transformation quality with the diversification strategy. At

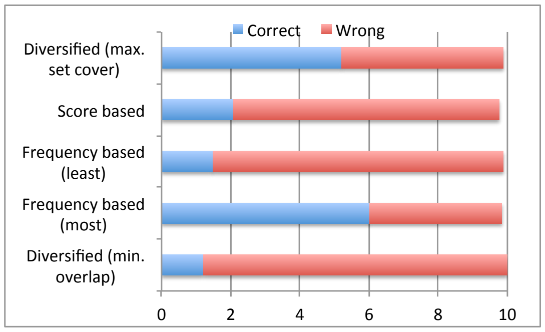


Fig. 8: The average ratio of correct and wrong transformations selected by the strategies for user-verification.

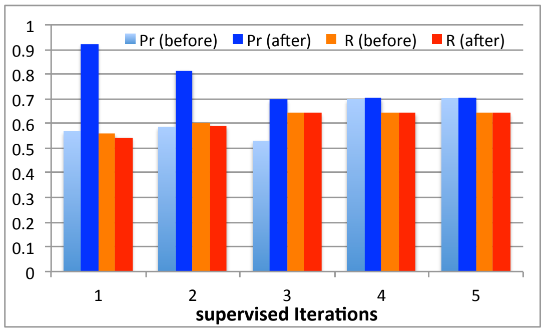


Fig. 9: Average precision and recall of `DataXFormer` before and after validating 10 examples per iteration.

each iteration the user validated 10 examples. The plot shows that the precision increases with the validation after each step. However, we notice a diminishing return in this gain, because we quickly reach enough information to make a decision. At the same time, the recall of the system slightly increases at each iteration. We achieved similarly results when running the same experiment with 20 validation examples per iteration.

VII. Related Work

There have been several attempts to tame the difficult task of transformation discovery [5], [6], [26]. Singh et al. [26] propose an approach for semantic string transformations, which is then implemented in MS Excel. While their approach combines syntactic and semantic transformations, it is limited to tasks where the tables containing the desired transformations are known a priori and are only very few, mostly 1 or 2 according to their benchmark. Similarly, Kandel et al. [6] support a language for the transformations involving manipulation of the data in the current relation. Our approach instead identifies transformations from a huge corpus of tables, requiring handling noise and ambiguity from these tables. Arasu et al. [5] address the problem of resolving abbreviations by string matching techniques. We discover this kind of transformations by explicitly looking them up in web tables.

Research on web tables has mostly focused on issues related to search, extraction, and integration [27], [28]. Web tables have been also regarded as a large repository for knowledge discovery. For example, InfoGather [29] addresses entity augmentation by searching for related attributes of given entities. It precomputes and indexes all inclusion dependencies to directly find overlapping tables. In our scenario with more than 100 million tables and usually missing schema, it is not feasible to discover and store all the possible inclusion dependencies. Instead, we discover overlapping columns at runtime. Furthermore, our system provides significant extensions through the various types of transformations.

Another line fo related research relates to providing information retrieval capabilities over structured databases. Most of the proposals [8]–[12] focus on efficiently generating candidate networks of joined tuples to form answers to a keyword query. In some of these systems, such as [16], [18], specialized indices or predefined foreign key-primary key relationships are used to prune the space of candidate results. Others assume the availability of context information [30] or meta-data and dependencies among keywords [17]. However, we are not interested in tuple networks, but rather in tables with columns covering most of the example transformations and the input values to be transformed. Since we do not have a complete knowledge of the underlying schema, our approach depends on instance matching (for web tables and KBs) and on discovering needed meta-data (for web forms).

Finally, we are the first to address the problem of discovering transformations for sets of entities; a task that is more challenging than simply discovering a unique target value. Identifying the entities in a set can be seen as an enumeration query in an open-world context. This problem has been tackled in the context of crowdsourcing [31] and truth discovery [25]. However, the crowdsourcing solution cannot be directly applied to our setting as they assume clean data. As we have shown in the experiments, the truth discovery algorithm makes strong assumptions on the prior distributions of the latent variables, while we do not make such assumption and we are still able to achieve better results.

VIII. Conclusion

This paper presents a full fledged system for transformation discovery. In particular, we focus on how our system fully exploit web tables to discover multi-column, non-functional, and indirect transformations. A comprehensive study based on 120 transformation tasks demonstrates the usefulness of our system. and shows the strength and weaknesses of each resource type for a transformation task. Future work requires reasoning on filtering transformations within a resource. Currently, we do not reject subsets of a table but only a table as a whole. An example-based approach combined with outlier detection and String pattern analysis could make the resource discovery more fine-granular. Furthermore, the discovery of transformation results through text analysis is an interesting and promising challenge.

References

[1] S. Kandel, A. Paepcke, J. M. Hellerstein, and J. Heer, "Enterprise data analysis and visualization: An interview study," *IEEE Trans. Vis. Comput. Graph.*, vol. 18, no. 12, pp. 2917–2926, 2012.

[2] 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.

[3] J. Morcos, Z. Abedjan, I. Ilyas, M. Stonebraker, P. Papotti, and M. Ouzzani, "DataXFormer: An interactive data transformation tool," in *SIGMOD*, 2015.

[4] X. Chu, J. Morcos, I. F. Ilyas, M. Ouzzani, P. Papotti, N. Tang, and Y. Ye, "Katara: A data cleaning system powered by knowledge bases and crowdsourcing," in *SIGMOD*, 2015, pp. 1247–1261.

[5] A. Arasu, S. Chaudhuri, and R. Kaushik, "Learning string transformations from examples," *PVLDB*, vol. 2, no. 1, pp. 514–525, 2009.

[6] S. Kandel, A. Paepcke, J. Hellerstein, and J. Heer, "Wrangler: Interactive visual specification of data transformation scripts," in *CHI*, 2011.

[7] B. Alexe, B. ten Cate, P. G. Kolaitis, and W. C. Tan, "Designing and refining schema mappings via data examples," in *SIGMOD*, 2011.

[8] S. Agrawal, S. Chaudhuri, and G. Das, "Dbxplorer: A system for keyword-based search over relational databases," in *ICDE*, 2002.

[9] 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*, 2002, pp. 1083–1086.

[10] V. Hristidis and Y. Papakonstantinou, "Discover: Keyword search in relational databases," in *VLDB*, 2002, pp. 670–681.

[11] L. Qian, M. J. Cafarella, and H. V. Jagadish, "Sample-driven schema mapping," in *SIGMOD*, 2012.

[12] Y. Shen, K. Chakrabarti, S. Chaudhuri, B. Ding, and L. Novik, "Discovering queries based on example tuples," in *SIGMOD*, 2014, pp. 493–504.

[13] P. A. Bernstein, J. Madhavan, and E. Rahm, "Generic schema matching, ten years later," *PVLDB*, vol. 4, no. 11, pp. 695–701, 2011.

[14] A. Das Sarma, X. Dong, and A. Halevy, "Bootstrapping pay-as-you-go data integration systems," in *SIGMOD*, 2008, pp. 861–874.

[15] L. Barbosa and J. Freire, "An adaptive crawler for locating hidden-web entry points," in *WWW*, 2007, pp. 441–450.

[16] A. Balmin, V. Hristidis, and Y. Papakonstantinou, "Objectrank: Authority-based keyword search in databases," in *VLDB*, 2004.

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

[18] J. Feng, G. Li, and J. Wang, "Finding top-k answers in keyword search over relational databases using tuple units," *TKDE*, vol. 23, no. 12, pp. 1781–1794, 2011.

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

[20] F. Murtagh, "Clustering in massive data sets," in *Handbook of massive data sets*. Springer, 2002, pp. 501–543.

[21] Z. Abedjan, J. Lorey, and F. Naumann, "Reconciling ontologies and the web of data," in *CIKM*, 2012, pp. 1532–1536.

[22] F. M. Suchanek, G. Kasneci, and G. Weikum, "Yago: A core of semantic knowledge," in *WWW*, 2007, pp. 697–706.

[23] C. Bizer, J. Lehmann, G. Kobilarov, S. Auer, C. Becker, R. Cyganiak, and S. Hellmann, "Dbpedia - a crystallization point for the web of data," *Web Semantics*, vol. 7, no. 3, pp. 154–165, Sep. 2009.

[24] J. Eberius, M. Thiele, K. Braunschweig, and W. Lehner, "Top-k entity augmentation using consistent set covering," in *SSDBM*, 2015.

[25] B. Zhao, B. I. P. Rubinstein, J. Gemmell, and J. Han, "A bayesian approach to discovering truth from conflicting sources for data integration," *PVLDB*, vol. 5, no. 6, pp. 550–561, 2012.

[26] R. Singh and S. Gulwani, "Learning semantic string transformations from examples," *PVLDB*, vol. 5, no. 8, pp. 740–751, 2012.

[27] M. J. Cafarella, A. Halevy, and N. Khoussainova, "Data integration for the relational web," *PVLDB*, vol. 2, no. 1, pp. 1090–1101, Aug. 2009.

[28] A. Das Sarma, L. Fang, N. Gupta, A. Halevy, H. Lee, F. Wu, R. Xin, and C. Yu, "Finding related tables," in *SIGMOD*, 2012, pp. 817–828.

[29] M. Yakout, K. Ganjam, K. Chakrabarti, and S. Chaudhuri, "Infogather: Entity augmentation and attribute discovery by holistic matching with web tables," in *SIGMOD*, 2012, pp. 97–108.

[30] R. Pimplikar and S. Sarawagi, "Answering table queries on the web using column keywords," *PVLDB*, vol. 5, no. 10, pp. 908–919, 2012.

[31] B. Trushkowsky, T. Kraska, M. J. Franklin, and P. Sarkar, "Crowdsourced enumeration queries," in *ICDE*, 2013, pp. 673–684.