Auto-Transform: Learning-to-Transform by Patterns

Zhongjun Jin*
University of Michigan
markjin@umich.edu

Yeye He
Microsoft Research
yeyehe@microsoft.com

Surajit Chauduri
Microsoft Research
surajitc@microsoft.com

ABSTRACT

Data Transformation is a long-standing problem in data management. Recent work adopts a "transform-by-example" (TBE) paradigm to infer transformation programs based on user-provided input/output examples, which greatly improves usability, and brought such features into mainstream software like Microsoft Excel, Power BI, and Trifacta.

While TBE is great progress, the need for users to provide paired input/output examples still poses limits on its applicability. In this work, we study an alternative that transforms data based on input/output *data patterns* only (without paired examples). We term this new paradigm *transform-by-patterns* (TBP). Specifically, we demonstrate that there is a rich class of transformations in TBP that can be "learned" from large collections of paired table columns. We show the proposed method can harvest such transformations across diverse domains and corpora (e.g., in different languages such as English, Chinese, Spanish, etc.). TBP transformations so obtained can be used in scenarios such as suggesting data-repairs in tables, or automating transformations in ETL pipelines. Extensive experiments on real data suggest that TBP outperforms existing methods on tasks such as data repairs, and is a promising direction for future research.

PVLDB Reference Format:
Zhongjun Jin, Yeye He, Surajit Chaurduri. Auto-Transform: Learning-to-Transform by Patterns. *PVLDB*, 13(11): 2368-2381, 2020.
DOI: https://doi.org/10.14778/3407790.3407831

1. INTRODUCTION

Data transformation is an important and long-standing problem in the data management literature [53]. For decades, *expert users* like developers or data engineers painstakingly write one-off programs/scripts to transform data from one format to another, across applications such as ETL [4, 26, 58], and data integration [14, 50].

*Work done at Microsoft Research.

Proceedings of the VLDB Endowment, Vol. 13, No. 11
ISSN 2150-8097.
DOI: https://doi.org/10.14778/3407790.3407831

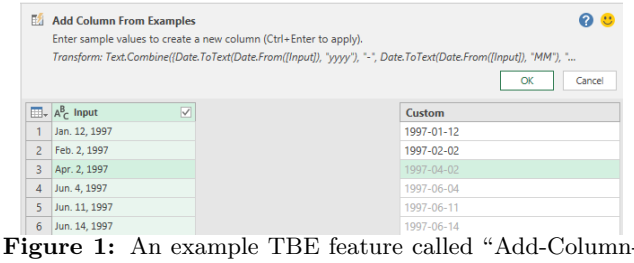


Figure 1: An example TBE feature called "Add-Column-From-Examples" in Microsoft Power BI. After typing two desired output examples in "Custom" column for the first two input values in the "Input" column, the system finds a consistent program (shown at the top), and gives a preview of output for remaining values (shown in gray).

Today, in a broader trend known as "self-service data preparation" [32, 56], *non-technical users* such as business analysts (e.g., in Excel or Tableau) increasingly need to manipulate data and perform tasks like data transformation. However, unlike expert users, these non-technical users lack the expertise to write programs. Democratizing data transformation for the non-technical users (e.g., without asking them to write code) has become increasingly important.

Transform-by-Example (TBE). In response to this demand, the "transform-by-example" (TBE) paradigm was developed for data transformation. In TBE, users provide a few *paired* input/output examples to demonstrate a desired transformation task. TBE systems would then search for programs consistent with all given examples, from a predefined space of candidate programs.

TBE has led to a fruitful line of research (e.g., [10, 31, 33, 36, 41, 42, 59, 68]), and has generated significant impact on commercial systems used by millions of people. For instance, TBE-like features are now available in Excel (using FlashFill [31])[1]; Power BI (using TDE [33])[2]; and Trifacta[3]. Google also announced plans to introduce TBE in Sheets[4].

Figure 1 shows an example TBE feature called "Add-Column-From-Examples" available in Microsoft Power BI[5]. This example spreadsheet has an "Input" column on the left with a list of date-time strings. In this case, a user

[1] https://www.microsoft.com/en-us/microsoft-365/blog/2012/08/09/flash-fill/
[2] http://powerbi.microsoft.com/en-us/blog/power-bi-desktop-june-feature-summary/#addColumn
[3] https://www.trifacta.com/blog/transform-by-example-your-data-cleaning-wish-is-our-command/
[4] https://cloud.google.com/blog/products/g-suite/connected-sheets-is-generally-available
[5] The same feature is also available in recent versions of Excel (e.g., Office 2019 and Office 365), under the "Data" tab.

S-timestamp	S-phone	S-coordinates
2019-12-23	(425) 882-8080	(38°57'N, 95°15'W)
2019-12-24	(425) 882-8080	(38°61'N, 95°21'W)
2019-12-23	(206) 876-1800	(39°19'N, 95°18'W)
2019-12-24	(206) 876-1800	(39°26'N, 95°23'W)
2019-12-23	(206) 903-8010	(39°42'N, 96°38'W)

R-timestamp	R-phone	R-coordinates
Nov. 16 2019	650-853-1300	N37°31′ W122°14′
Nov. 17 2019	650-853-1300	N37°18′ W122°19′
Nov. 16 2019	425-421-1225	N37°48′ W122°17′
Nov. 17 2019	425-421-1225	N37°60′ W123°08′
Nov. 16 2019	650-253-0827	N37°01′ W123°72′

Figure 2: Two tables R and S with schema (time-stamps, phone-number, geo-coordinates). Integrating the two would require values to be reformatted using transformations.

invokes the TBE feature, and enters two output examples (`1997-01-12` and `1997-02-02`) in the "Custom" column on the right, to demonstrates a desired transformation. In response to user input, the system synthesizes a transformation program consistent with the two given input/output examples, which is shown at the top of the figure (this program invokes a total of 7 functions, including `Text.Combine, Date.ToText`, etc.). Furthermore, a preview of remaining output values is shown in gray (beneath user-provided examples), which helps users to verify the correctness of the suggested transformation.

Transform-by-Pattern (TBP). The by-example TBE paradigm is clearly an excellent fit for Excel-like spreadsheet environments. As we will see below, however, in other settings it may not be as easy to invoke TBE, for it can be hard for users to identify columns requiring transformations, and then provide paired input/output examples. We in this work propose an alternative Transform-by-Pattern (TBP) paradigm to complement the TBE approach, which can proactively suggest relevant transformations based only on *input/output data patterns* (with no paired examples).

More concretely, each TBP program is a triple (P_s, P_t, T), where P_s and P_t are data "patterns" (e.g., in regex) describing the source and target column, for which the corresponding program T is applicable.

Table 1 shows a list of example TBP programs (we will discuss how to harvest them in detail). Each row here is a TBP program that consists of a triple (P_s, P_t, T). For the TBP program labeled as `TBP-1` in the first row, its source pattern P_s is: "`<letter>{3}. <digit>{2}, <digit>{4}`" and target pattern P_t is: "`<digit>{4}-<digit>{2}-<digit>{2}`". Note that these two patterns can be used to describe the example TBE case shown in Figure 1; the corresponding transformation program (shown at the top of Figure 1) can be "memorized" in the last column T of Table 1 (omitted in the table in the interest of space).

In the following, we use two concrete applications, Auto-Unify and Auto-Repair, to demonstrate that such TBP programs can enable scenarios complementary to TBE. We emphasize that TBP is *not* meant to replace the general-purpose TBE, especially in spreadsheet settings where users can easily identify target output and enter examples.

TBP for "Auto-Unify". Data transformation is often required in applications like ETL and data integration, where data of different formats from multiple sources need to be unified and standardized.

Figure 2 shows two example tables denoted by R and S, both containing telemetry data of the form: (`time-stamp`, `cellular-device-numbers`, `geo-coordinates`). As is often the case in the real world, R and S are formatted differently (e.g., the telemetry may be generated by different types of devices, or different versions of programs), and need to be integrated, which is a common task in ETL [26, 44].

Today, data engineers need to first identify such issues like in Figure 2 (a time-consuming task when there are many such feeds and columns). They would then write ad-hoc transformation scripts, in order to unify each pair of incompatible data columns.

We argue that armed with a repository of TBP programs like in Table 1, the task of identifying and addressing aforementioned issues can be partially automated. Specifically, given that `R-timestamp` and `S-timestamp` need to be merged, based on the patterns of values in these two columns, we can suggest `TBP-1` in Table 1 to be used, because its source pattern P_s = "`<letter>{3}. <digit>{2}, <digit>{4}`" and target pattern P_t = "`<digit>{4}-<digit>{2}-<digit>{2}`" match with `R-timestamp` and `S-timestamp`, respectively. This allows us to proactively suggest the corresponding T to perform this transformation.

Similarly, the patterns P_s and P_t in `TBP-2` and `TBP-3` from Table 1 would match with column-pairs (`S-phone`, `R-phone`) and (`S-coordinates`, `R-coordinates`) in Figure 2, respectively, suggesting two additional transformations that can be performed. It should be noted that TBE typically requires *paired* examples and would not apply here.

TBP for "Auto-Repair". As an additional example application, we show that TBP can also help to identify and fix inconsistent data values in tables. Figure 3 shows real data quality issues in Wikipedia tables that are identified and fixed by TBP programs produced in this work.

For instance, in Figure 3(a), using TBP we can detect that values in the `Date` column have two distinct patterns: "`<digit>{4}-<digit>{2}-<digit>{2}`" (e.g., "`1997-06-04`") as well as "`<letter>+ <digit>{2}, <digit>{4}`" ("`January 12, 1997`"). Since these two patterns match with P_s and P_t of a TBP program in Table 1, it likely indicates data inconsistency. With TBP, we could bring these two groups of values to users attention, and propose fixes by applying the corresponding T (e.g., transforming "`1997-06-04`" to "`June 4, 1997`").

We note that the TBP framework is general and applies to diverse types of transformations, including data in different languages (e.g., Spanish, Chinese, etc.), and data in different domains (e.g., chemical, financial, etc.). For example, some of the cases in Figure 3 require transformations in languages other than English, such as Figure 3(e) (fixable by `TBP-15`), and Figure 3(l) (fixable by `TBP-16`), etc. These are all real TBP programs harvested from different table corpora (e.g., Wikipedia tables in different languages). Our evaluation suggests that these TBP programs can detect and fix thousands of real issues across different languages.

For non-technical users working on spreadsheet data (e.g., in Microsoft Excel or Tableau), TBP makes it possible to automatically flag and repair a subclass of data format issues. We note that TBP once again complements traditional TBE approaches, which would require explicit paired-examples in order to suggest transformations.

In short, TBP can program a rich class of transformations, creating opportunities to simplify data transformation in applications such as Auto-Repair and Auto-Unify.

Table 1: An example repository of TBP programs (P_s, P_t, T), where each line is a TBP program. The first three programs can be used to auto-unify the two tables shown in Figure 2.

TBP-id	Source-pattern (P_s)	Target-pattern (P_t)	(T)
TBP-1	<letter>{3}. <digit>{2}, <digit>{4}	<digit>{4}-<digit>{2}-<digit>{2}	...
TBP-2	(<digit>{3}) <digit>{3}-<digit>{4}	<letter>{3}-<digit>{3}-<digit>{4}	...
TBP-3	(<digit>+°<num>'<letter>{1}, <digit>+°<num>'<letter>{1})	<letter>{1}<digit>+°<num>' <letter>{1}<digit>+°<num>'	...
...	...	...	...
TBP-7	<digit>{4}/<digit>{2}/<digit>{2}	<letter>{3} <digit>{2}	...
TBP-8	<num> kg	<num> lb	...
TBP-9	<num> lb	<num> lb <num> oz	...
...	...	...	...
TBP-15	<num> kg	<num>公斤	...
TBP-16	<letter>+ de <digit>{4}	<digit>{4}	...
...	...	...	...

Table 2: Example table with (C, C', T) triples, where (C, C') are paired columns, and T is a synthesized program that can transform C to C'. The first triple `CCT-1` corresponds to the column-pair ("`Born`", "`Date of birth`") in Figure 4, with an inferred program in Listing 1. `CCT-4` shows another pair of columns with similar data format and an identical program. Not all column-pairs have programmatic relationships, such as `CCT-9`, leading to an empty program.

CCT-id	Input-column (C)	Output-column (C')	Program (T)
CCT-1	(C_1) "`Born`" = {"`02/22/1732`", "`10/30/1735`", ... }	(C'_1) "`Date of birth`" = {"`February 22, 1732`", ... }	Listing 1
CCT-2	(C_2) "`Date of birth`" = {"`February 22, 1732`", ... }	(C'_2) "`Born`" = {"`02/22/1732`", "`10/30/1735`", ... }	...
CCT-3	(C_3) "`Died`" = {"`02/14/1799`", "`07/04/1826`", ... }	(C'_3) "`Date of birth`" = {"`February 22, 1732`", ... }	...
CCT-4	(C_4) "`Date`" = {"`11/01/2019`", "`12/01/2019`", ... }	(C'_4) "`Date-2`" = {"`November 01, 2019`", ... }	Listing 1
...	...	...	...
CCT-9	(C_9) "`Name`" = {"`Washington, George`", "`Adam, John`", ... }	(C'_9) "`Date of birth`" = {"`February 22, 1732`", ... }	$\emptyset$
...	...	...	...

"Learned" TBP programs from TBE query logs. Given the benefit of TBP, we set out to harvest such programs at scale (as manually curating them would not scale).

One possible approach is to leverage the "query-logs" of a TBE system. This is analogous to search engines like Google and Bing, which have long used their query logs containing (`keyword-query`, `user-clicked-document`) to improve search relevance. We argue that the same is true for TBE systems – specifically, since we have developed TDE [33] and deployed a version of the system as an Excel add-in, we are able to collect telemetry of TBE tasks submitted by Excel users. We should emphasize that we could *not* log user data in any form due to legal and compliance reasons – we only collect high-level statistics such as whether a top-ranked transformation program suggested by TDE is accepted. Hypothetically, imagine that we could fully log users input/output data sets, then like search engines we could leverage the logs to identify common (`input-data-pattern`, `output-data-pattern`, `program`) triples that are likely good TBP programs.

Because we are not able to obtain detailed logs in spreadsheet programs, in this work we develop alternative approaches to harvest TBP programs.

"Learned" TBP programs from tables. In this work we propose to harvest TBP transformations from a large collection of tables. Specifically, we develop techniques to automatically "link" together table columns with related content, from which we can exploit content redundancy to "learn" common transformations.

Figure 4 shows 6 example web tables about US presidents. We develop techniques to link them together at a row-level – e.g., the first row of each table corresponds to "`George Washington`" and will link/join. After rows are linked, we can pair columns together "as if" they are input/output columns, to see if any transformation can be learned using TBE – for example, the "`Born`" column {"`02/22/1732`", "`10/30/1735`", ...} in T_1 can be paired with the "`Date of birth`" column {"`February 22, 1732`", "`October 30, 1735`", ...} from T_2, etc. Table 2 shows this column-pair, in row `CCT-1`, as well as many other column pairs so produced.These column pairs are then fed into a TBE system (in our case, TDE [33]) to learn possible transformation programs, which are stored in the last column of the table. Notice that given 6 different date-formats used by 6 tables for date-of-birth in Figure 4, we can already construct a total of $2\binom{6}{2} = 30$ distinct pairs of formats and their corresponding transformations, which are all validate TBP programs.

Figure 5 shows another group of 5 tables from Wikipedia, each of which has a table for US presidents but in different languages. We develop methods to again automatically link rows between these tables, and then construct column-pairs for TBE systems to learn possible transformation programs across different languages (e.g., from "`April 30, 1789`" to "`30 de abril de 1789`").

By analyzing many such (Input-column, Output-column, Transformation-program) triples in Table 2, we can identify programs that are used repeatedly across the corpus – for example, the same program (labeled as Listing 1 in Figure 2) is being used by column-pair `CCT-1`, `CCT-4` and many others, suggesting that this is likely a good TBP program. In this work, we develop methods to construct a large "transformation graph", to reason about the goodness of TBP programs in a global manner. TBP programs so produced can then be used to enable applications like Auto-Repair.

Inter-operability of structured data. TBP is one step toward achieving *inter-operability* of tabular data. We note that by "lifting" data values from a "string" space into a "program/code" space using TBP, values become inter-operable (via programs). This is analogous to knowledge-bases used in search engines, which also "lift" strings into "entities" for richer experiences (e.g., knowledge cards and related entities as opposed to 10 blue links). TBP can similarly light up new experiences for tabular data like Auto-Repair, and is a useful step toward inter-operability.

(a) EN-Wiki: Dates (b) EN-Wiki: Currency values (c) EN-wiki:time (d) EN-Wiki: Date

(e) ZH-Wiki: Units (f) ZH-Wiki: Ordinals (g) ZH-Wiki: Date (h) JA-Wiki: Year

(i) JA-Wiki: Date (j) ES-Wiki: Numbers (k) ES-Wiki: Numbers (l) ES-Wiki: Date

Figure 3: Auto-Repair: Real quality issues (in red boxes) from Wikipedia tables that are fixable by TBP programs. Note that the examples span different languages (English, Chinese, Japanese, Spanish, etc.)

2. SYSTEM ARCHITECTURE

Figure 6 gives a high-level overview of the architecture of our system. There are three main components, which are all offline processing steps. The first component takes a large corpus of tables (e.g., web tables or enterprise spreadsheets), find related tables, link/join records across tables (like shown in Figure 4 and Figure 5), to produce paired columns (C, C') like in Table 2 (Section 3).

The second component uses paired columns (C, C') as if they are input/output columns in a transformation task, and invokes TBE to find possible transformation T consistent with all examples in (C, C'). If TBE synthesizes such a T, the (C, C', T) triple is populated in Table 2 (Section 4).

In the last stage, we analyze (C, C', T) triples in Table 2 in a global manner, in order to identify TBP programs that are both commonly-used and highly-accurate. We formulate an automated approach to harvest such programs, as well as a human-curated variant that can leverage human labels effectively (Section 5).

We now discuss each component in turn.

3. PAIR COLUMNS WITH LINKED ROWS

In this section, we discuss the first part of our system, which takes a large collection of tables $\mathbf{T}$ as input, and produces pairs of columns that are linked row-by-row. In this section, we discuss 3 different ways to achieve this in turn, using a corpus of over 100M web tables [18][6]

3.1 Pair Columns by Search Engine

Our first approach leverages search engines, utilizing the observation that pages returned for the same keyword query often contain related tables. We perform 3 steps here: pairing tables, linking rows, and pairing columns.

Pairing tables. We take the query-logs of a commercial search engine, and first use a production classifier [18] to select queries known as "table-intent queries" [18], which are data-seeking queries such as "`list of us presidents`", "`list of national parks`", "`list of chemical elements`", etc. We obtain a total of 16M table-intent queries, denoted by $\mathbf{Q}$.

For each query $q \in \mathbf{Q}$, we retrieve all web tables in the top-20 pages returned by the search engine, denoted by T_q, which contains tables related to query q. For example, tables in Figure 4 are all retrieved for the query "`list of us presidents`". We can then pair such tables in T_q to produce table-pairs $P_{\mathbf{Q}} = \{(T, T') | T \in T_q, T' \in T_q, T \neq T', q \in \mathbf{Q}\}$.

Linking rows. Recall that in order to utilize TBE to generate programs, we need *paired* input/output examples.

[6]Similar web-table data sets are publicly available in [2, 8].

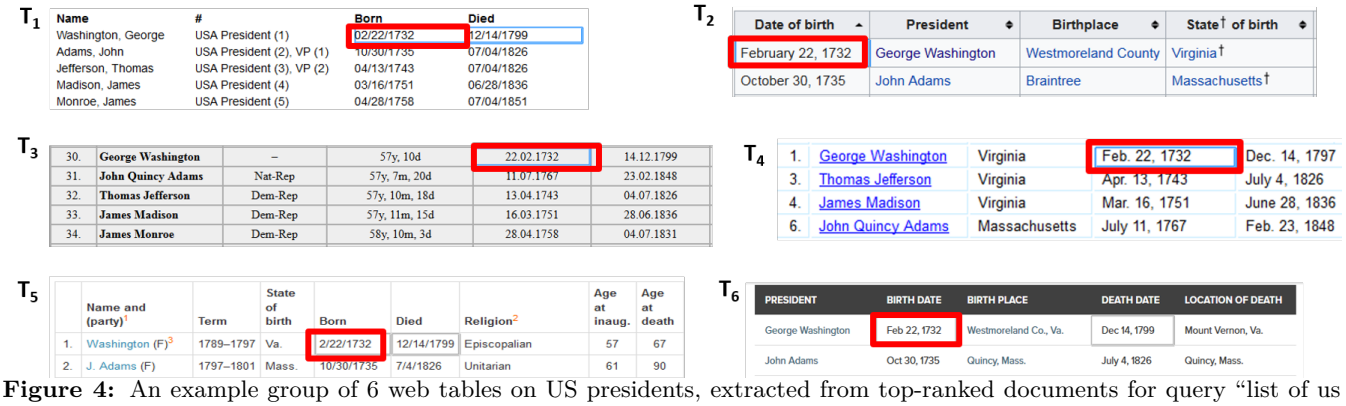


Figure 4: An example group of 6 web tables on US presidents, extracted from top-ranked documents for query "list of us presidents". Note that the same date-of-birth information is being represented in 6 different formats, which can be used as input/output examples for TBE to learn common TBP transformations.

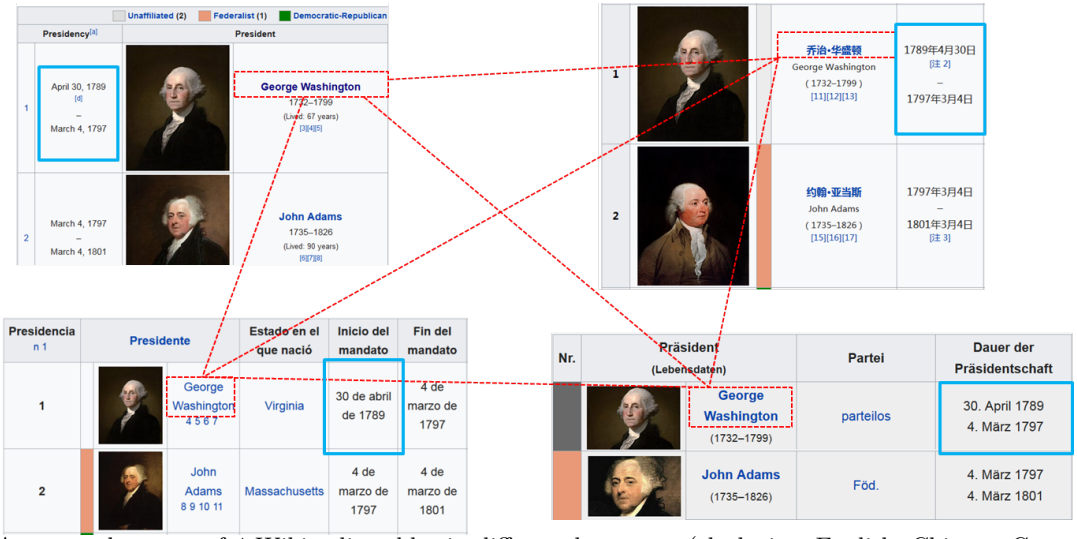


Figure 5: An example group of 4 Wikipedia tables in different languages (clockwise: English, Chinese, German, Spanish) that we can link at a row-level (using Wiki inter-language links for pages with the same content). Note that the "date-in-office" is being represented in different languages across 4 tables, providing examples to learn such transformations.

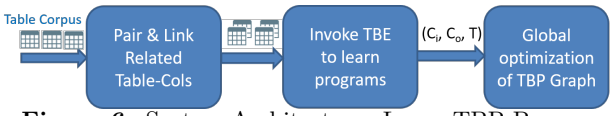


Figure 6: System Architecture: Learn TBP Programs.

So for a given pair $(T, T') \in P_{\mathbf{Q}}$, we additionally need to find row-level "links" between T and T' (e.g., the first row of T_1 in Figure 4 corresponds to the first row of T_2, etc.).

In an ideal setting, such row-level links can be obtained by equi-joins on key-columns. However, in practice equi-joins typically fail, because values are often coded/formatted differently between tables in the wild – e.g., names of presidents are formatted differently as shown in Figure 4.

To account for syntactic variations in the key-columns, we leverage an existing "auto-join" system [68][7] to automatically identify join relationships. Specifically, given $(T, T') \in P_{\mathbf{Q}}$, we take two left-most non-numeric columns from T and T' (which likely include key columns), and invoke the "auto-join" system to find possible joins.

EXAMPLE 1. For T_1, T_2 in Figure 4, we use [68] to automatically infer a join-program J, which performs the following operations on the "`Name`" column of T_1: (1) splitting names like "`Washington, George`" by comma (producing an array with two elements like ["`Washington`", "`George`"]); (2) concatenating the second element with the first element using a space (producing values like "`George Washington`").

Note that applying J on the "`Name`" column in T_1 produces values like {"`George Washington`", "`John Adams`", ...}, which precisely match the key values in T_2, such that an "equi-join" can now be performed to link the two tables together. We consider this J to be reliable, as most rows can be joined 1:1 using J.

The auto-join approach allows us to link rows together between other table-pairs in Figure 4 similarly. We find this approach produces substantially more links than equi-join, which are also more accurate than fuzzy-join (because it uses precise transformations). More details of this step can

[7] A variant of this system is publicly available in Azure ML Data Prep: `https://docs.microsoft.com/en-us/python/api/azureml-dataprep/azureml.dataprep.api.builders.joinbuilder`

be found in [68], which we omit here in the interest of space.[8]

Pairing Columns. For each table pair $(T, T') \in P_{\mathbf{Q}}$ that is now linked at a row-level, we enumerate pairs of columns in (T, T'), written as $C_{\mathbf{Q}} = \{(C, C') | (T, T') \in P_{\mathbf{Q}}, C \in T, C' \in T'\}$. This produces a total of 15M column pairs, which will later be used as input/output examples for TBE.

EXAMPLE 2. For T_1, T_2 in Figure 4, we produce (C, C') column pairs such as (`Name`, `Date-of-birth`), (`#`, `Date-of-birth`), (`Born`, `Date-of-birth`), etc., which would populate Table 2. Observe that values in C and C' are ordered based on row-level links from the previous step, and not all (C, C') column-pairs have programmatic relationships (e.g., `CCT-9`).

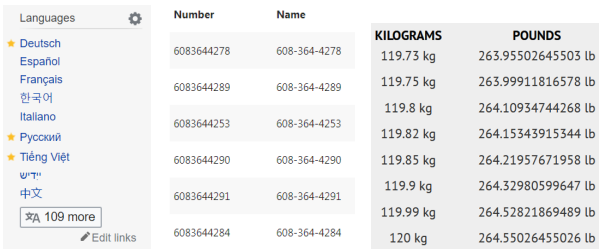
3.2 Pair Columns by Wikipedia Links

Our second approach pairs columns leveraging intra-Wiki cross-language links.

Pairing Tables. Wiki pages have extensive intra-Wiki links pointing to other related Wiki pages. A special form of this is the cross-language links. Specifically, each Wiki page p has a list of links on the left side-bar pointing to Wiki pages with the same content as p but written in other languages (e.g., Figure 7(a) shows the cross-language links on page "`list of us presidents`" that point to similar pages written in 109 other languages).

We parse Wiki pages from a recent crawl, and identify cross-language links to produce $L_{\text{link}} = \{(p, p')\}$ that records all pairs of pages p, p' linked by cross-language links, from which we can again identify pairs of related tables as: $P_{\text{wiki}} = \{(T, T') | T \in p, T' \in p', (p, p') \in L_{\text{link}}\}$. Each pair (T, T') so produced will likely have related content in different languages, like shown in Figure 5.

Linking Rows. Given table pairs $(T, T') \in P_{\text{wiki}}$, we leverage cross-language links for a second time to identify row-level links between (T, T'). Specifically, notice that for each table in a given language (e.g., English, Chinese, etc.) in Figure 5, the president names are all blue links pointing to Wiki entity pages of these presidents in that same language – e.g., the first row of the top two tables in Figure 5 links to the Wiki entity page of "`George Washington`" in English and Chinese, respectively, denoted as p_{gw}^{en} and p_{gw}^{zh}. Recall that these two pages p_{gw}^{en} and p_{gw}^{zh} are again linked by cross-language links, or $(p_{gw}^{en}, p_{gw}^{zh}) \in L_{\text{link}}$, this allows us to determine that these two first rows should be linked together. Links for other rows can be produced similarly, as shown with dotted lines in Figure 5.



(a) Wiki-links: other language pairs (same table). (b) Example col (same table). (c) Example col pairs

Figure 7: Example web pages.

[8]We note that this row-wise linking step can be seen as a form of unsupervised entity-resolution, where other techniques like [34, 47, 62, 67] may also be used.

We currently extract pages in 12 popular languages (`en, es, de, fr, ja, ka, it, zh, vi, pt, hi, ar, ru`), and focus on language-pairs involving English (e.g., `en-es`, `en-de`, etc.). This produces 3M table pairs across languages.

Pairing Columns. From P_{wiki}, columns paired can again be enumerated as $C_{\text{wiki}} = \{(C, C') | (T, T') \in P_{\text{wiki}}, C \in T, C' \in T'\}$, with values in (C, C') paired based on row-level links. Examples of C_{wiki} include pairs of different "`time-in-office`" columns shown in Figure 5.

3.3 Pair Columns Within Individual Tables

Lastly, observing that column pairs within individual tables can also have programmatic relationships (e.g., Figure 7(b) and 7(c)), we produce single-table column pairs as $C_{\mathbf{T}} = \{(C, C') | T \in \mathbf{T}, C \in T, C' \in T, C \neq C'\}$, where $\mathbf{T}$ is the corpus of all web tables. Note that such column-pairs are from the same table and are thus already linked at a row-level. This generates over 1 billion column pairs.

4. GENERATE TBE PROGRAMS

Given the paired-columns from the three approaches in Section 3, denoted by $\mathbf{C} = C_{\mathbf{Q}} \bigcup C_{\text{wiki}} \bigcup C_{\mathbf{T}}$, we take column-pairs $(C, C') \in \mathbf{C}$ and invoke TBE to test if there is any programmatic relationships.

In this work, we use the TDE system [33] as our TBE engine. Unlike other TBE systems, TDE can synthesize complex programs using external functions from GitHub.

EXAMPLE 3. Consider column pair ("`Born`"= {"`02/22/1732`", "`10/30/1735`", ...}, "`Date of birth`" = {"`February 22, 1732`", "`October 30, 1735`", ...}), shown in the `CCT-1` row of Table 2. (this column-pair is taken from Figure 4).

Figure 8 illustrates how TDE synthesizes a program that can transform each value in the column "`Born`" (on the left) to exactly match a corresponding value in the column "`Date of birth`" (on the right).

In this example, TDE identifies a function in its index called `DateTime.Parse(String)`[9] as a promising candidate, and invokes the function with each input value in "`Born`" as parameter (e.g., `DateTime.Parse`("`02/22/1732`")).

For each input value, invoking `DateTime.Parse(String)` produces a `DateTime` object, which has attributes such as `Year`, `Month`, `Day` that are populated with relevant values. Conceptually, these values can be organized as an intermediate table as shown in Figure 8.

From this table, we can leverage program-synthesis techniques to "piece-together" relevant parts, so that the output on each input value can exactly match its target. Let `ret` be the object returned from invoking `DateTime.Parse(String)`. A synthesized program T performs the following steps:
(1): Take `ret.Month-Str`;
(2): Concatenate (1) with an empty-space "␣";
(3): Concatenate (2) with `ret.Day`;
(4): Concatenate (3) with "`, ␣`";
(5): Concatenate (4) with `ret.Year`.

This synthesized program is shown below, and can be verified to transform values in "`Born`" to ones in "`Date of birth`".

Listing 1: Synthesized program for the example in Figure 8

```
public string T(string input)
{  var ret = System.DateTime.Parse(input);
   return ret.Month-Str + ‘‘ ’’ + ret.Day + ‘‘, ’’  + ret.Year;
}
```

[9]https://docs.microsoft.com/en-us/dotnet/api/system.datetime.parse

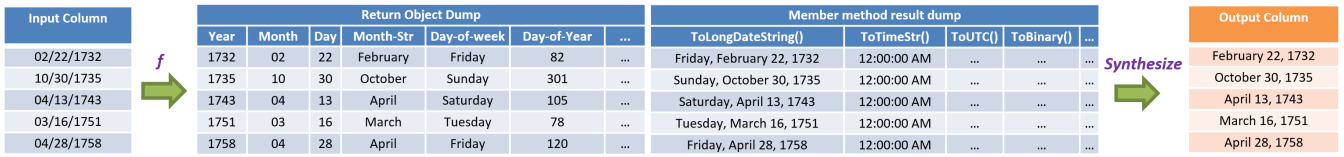


Figure 8: Example Invocation of f = `DateTime.Parse()` on each value from the input column (left). Each invocation returns a `DateTime` object, whose attribute-values can be conceptually represented as the tables shown in the middle. Programs can be dynamically synthesized to produce the exact values in the output column (right).

For each column pair (C, C') from Table 2 where a program T can be learned, we populate T in the last column of Table 2. For example, Listing 1 is populated for `CCT-1` and `CCT-4`. We refer to this collection of (C, C', T) triple as $\mathbf{T}_{\text{CCT}} = \{(C, C', T)\}$.

5. DISCOVER TBP PROGRAMS

So far we produced (C, C', T) triples in $\mathbf{T}_{\text{CCT}}$ like shown in Table 2. Our next step is to conflate related CCT triples, so that useful TBP programs in the form of (P_s, P_t, T) like in Table 1 can stand out.

The first challenge we address in this section, is to generate suitable patterns P_s and P_t from C and C', respectively.

5.1 Generate Patterns for Data Columns

In this work, we choose to use regex-like patterns to instantiate both P_s and P_t in TBP programs. Note that other ways to generate pattern-signatures and group related data columns, e.g., via semantic type-detection [64, 66], are also possible and are interesting directions to extend TBP.

Since generating regex-patterns given a data column C has been extensively studied in the data profiling literature (e.g., [28, 54]), it is not the focus of this work. We perform pattern-profiling using an in-house implementation that enumerates *all* possible regex-like patterns for a given column C, denoted by $\mathbf{P}(C)$.[10]

Given all possible patterns $\mathbf{P}(C)$ for a column C, our observation is that not all patterns in $\mathbf{P}(C)$ are equally suitable for TBP, as illustrated in the following example.

Example 4. Consider the row `CCT-1`, with a triple (C_1: {"`02/22/1732`", "`10/30/1735`"}, C_2: {"`February 22, 1732`", "`October 30, 1735`"}, Listing 1) in Table 2. Given C_1, the possible patterns that can be generated in $\mathbf{P}(C_1)$ include: P_{1a} = "`<digit>{2}/<digit>{2}/17<digit>{2}`" (with a constant "`17`" since all years in C_1 starts with "`17`"); a more general P_{1b} ="`<digit>{2}/<digit>{2}/<digit>{4}`"; or a even more general P_{1c} ="`<num><symbol><num><symbol><num>`", etc. Here "`<digit>`" stands for `[0-9]`, "`<num>`" for both integer and floating-numbers, "`<symbol>`" for any punctuation, and "`<letter>`" for `[a-zA-Z]`.

As humans, we can intuitively see that the ideal way to generalize `CCT-1` into a TBP program, is to use the second option P_{1b} = "`<digit>{2}/<digit>{2}/<digit>{4}`" to "describe" column C_1, because it would "match" other similar columns to which the same transformation in Listing 1 also applies (e.g., column C_4 = {"`11/01/2019`", "`12/01/2019`" } of `CCT-4` in Table 2). In comparison, using a less general pattern like P_{1a} to describe C_1 would lead to reduced applicability. At the same time, using a more general pattern like P_{1c} = "`<num><symbol><num><symbol><num>`" would make Listing 1 trigger on irrelevant columns (e.g., phone numbers like `425-880-8080`), thus producing false-positives.

The upshot is that only $P_{1b} \in \mathbf{P}(C_1)$ generalizes values in C_1 into the right level for TBP, which strikes the right balance between generality and accuracy of matches in the context of this TBP transformation.

Similarly, while many patterns can be generated for C'_1 in `CCT-1`, only the pattern $P_{2b} \in \mathbf{P}(C'_1)$, "`<letter>+ <digit>{2}, <digit>{4}`", is a more suitable choice for this TBP program.

Example 4 illustrates that given a triple (C, C', T), there are many possible patterns $\mathbf{P}(C)$ and $\mathbf{P}(C')$, from which we need to pick suitable choices. Intuitively, we want to pick patterns $P \in \mathbf{P}(C)$ and $P' \in \mathbf{P}(C')$ to balance two conflicting objectives: (1) High "coverage", or we should pick patterns that are general to ensure that the resulting TBP is broadly applicable; and (2) High "accuracy", or we should pick patterns that are not overly general to make the resulting TBP trigger on irrelevant column-pairs.

Our observation is that while it is hard to know what P and P' to pick by only looking at one (C, C', T) triple, such decisions would become obvious when looking at a large collection of triples $\mathbf{T}_{\text{CCT}}$ (from Section 4).

In the following, we will first formally define *coverage* and *accuracy* of a TBP program using $\mathbf{T}_{\text{CCT}}$, and then illustrate these notions using concrete examples.

Definition 1. The estimated *coverage* of a TBP program (P, P', T) on $\mathbf{T}_{\text{CCT}}$, denoted by $Cov(P, P', T)$, is defined as:

$$Cov(P, P', T) = |\{(C, C', T') | (C, C', T') \in \mathbf{T}_{\text{CCT}}, P \in \mathbf{P}(C), P' \in \mathbf{P}(C'), T = T'\}| \quad (1)$$

$Cov(P, P', T)$ identifies the number of triples in $\mathbf{T}_{\text{CCT}}$, where P matches C ($P \in \mathbf{P}(C)$), P' matches C' ($P' \in \mathbf{P}(C')$), and T is applicable.

Definition 2. The estimated *accuracy* of a TBP program (P, P', T) on $\mathbf{T}_{\text{CCT}}$, denoted by $Acc(P, P', T)$, is defined as:

$$Acc(P, P', T) = \frac{Cov(P, P', T)}{|\{(C, C', T) | (C, C', T) \in \mathbf{T}_{\text{CCT}}, P \in \mathbf{P}(C), P' \in \mathbf{P}(C')\}|} \quad (2)$$

$Acc(P, P', T)$ measures the fraction of column pairs matching P and P', for which T is applicable.

We use the following example to illustrate how *coverage* and *accuracy* of a TBP program (P, P', T) can be estimated using a global analysis of $\mathbf{T}_{\text{CCT}}$.

Example 5. Continue with Example 4. For `CCT-1`, suppose we pick $P_{1b} \in \mathbf{P}(C_1)$ and $P_{2b} \in \mathbf{P}(C'_1)$, this produces a candidate TBP program $\text{TBP}_{1b} = (P_{1b}, P_{2b}$, Listing-1). If we look across triples in $\mathbf{T}_{\text{CCT}}$, we find additional "evidence" for which TBP_{1b} is applicable. For example, in `CCT-4`, C_4 and C'_4 are consistent with P_{1b} and P_{2b}, respectively, and furthermore T in `CCT-4` is identical to Listing-1. Intuitively, if we find many such triples in $\mathbf{T}_{\text{CCT}}$, we can reason TBP_{1b} to be high coverage.

Suppose we find a total of 800 triples $(C, C', T) \in \mathbf{T}_{\text{CCT}}$, for which P_{1b} matches C, and P_{2b} matches C'. Intuitively these (C, C') are column-pairs on which TBP_{1b} can trigger. Suppose we find that 600 triples out of the 800 have the same program Listing-1. This would indicate that TBP_{1b} has a good "coverage" of 600 column-pairs, using Equation (1), as

[10] Different pattern languages can be used here – we consider the exact choice of patterns as orthogonal to TBP.

well as a good "accuracy" of $\frac{600}{800}$, using Equation (2), making TBP_{1b} a desirable TBP candidate.

A second candidate $\text{TBP}_{1a} = (P_{1a}, P_{2b}$, Listing-1) uses a less general P_{1a} = "`<digit>{2}/<digit>{2}/17<digit>{2}`" from Example 4. Suppose we find 10 triples in $\mathbf{T}_{\text{CCT}}$ for which TBP_{1a} is applicable (because P_{1a} has a restrictive prefix "`17`"). This makes TBP_{1a} low coverage and less desirable.

A third candidate $\text{TBP}_{1c} = (P_{1c}, P_{2b}$, Listing-1) uses a more general P_{1c} = "`<num><symbol><num><symbol><num>`". The two patterns P_{1c} and P_{2b} would match 10000 column pairs (C, C') in $\mathbf{T}_{\text{CCT}}$, within which Listing-1 is applicable to only 600. This translates to a low accuracy score of $\frac{600}{10000}$, which makes TBP_{1c} less desirable.

5.2 A graph-based analysis of TBP programs

Our coverage and accuracy scores defined so far would reliably reflect the quality of TBP programs, if we could actually "log" real input-output columns users submit to a TBE system, and use these real column-pairs/transformation-tasks to estimate the quality of TBP programs.

However, because logging user data is not allowed in spreadsheet settings, we rely on columns paired synthetically as a "proxy" of real transformation tasks, many of which may have no programmatic relationships. As a result, our estimate of accuracy in Equation (2) can be inaccurate.

As an example, when columns are auto-paired in Figure 4, note that the "`Died`" column in T_1 and the "`Date of birth`" column in T_2 would also be paired, producing a triple `CCT-3` shown in Table 2. Since there is no relationship for the two columns in `CCT-3`, no programs can be learned. However, since the the columns in `CCT-3` have identical patterns as those in `CCT-1`, the estimated accuracy of TBP programs for `CCT-1` (e.g, TBP_{1b} in Example 5) would be lowered, because `CCT-3` is (incorrectly) included in the denominator of Equation (2) when accuracy is computed.

We should note that simply pruning away triples with no learned-program in $\mathbf{T}_{\text{CCT}}$ would also not work as it overestimates the true accuracy of programs – the TDE engine we leverage uses powerful external functions, which can occasionally "over-fit" complicated programs on unrelated column-pairs, which would then be suggested.

Let $\mathbf{T}_{\text{PPT}} = \{(P, P', T) | (C, C', T) \in \mathbf{T}_{\text{CCT}}, P \in \mathbf{P}(C), P' \in \mathbf{P}(C')\}$, be the space of possible candidate programs we enumerate. Given the large space of candidates in $\mathbf{T}_{\text{PPT}}$, the coverage and accuracy measures help us to prune unpromising TBP candidates (e.g., coverage < 5, accuracy < 0.1). From the remaining candidates, we propose to leverage program interactions to find good TBP programs, by analyzing all candidate TBP programs in a global TBP graph G.

Definition 3. We model TBP transformations in $\mathbf{T}_{\text{PPT}}$ using a directed graph $G = (V, E)$, where each pattern P corresponds to a vertex $V_P \in V$, and each candidate program $(P, P', T) \in \mathbf{T}_{\text{PPT}}$ corresponds to a directed edge $E_{PP'T} \in E$ that connects vertex V_P to $V_{P'}$.

We note that this is a directed graph because TBP programs are directional (e.g., T typically converts data in pattern P to pattern P', but not in the other direction).

Example 6. Figure 9 shows an example TBP graph with sample vertices and edges. Each vertex here corresponds to a pattern, and each edge is a candidate TBP program (some edges are omitted to avoid clutter).

We also note that in general there can be multiple edges between vertices (making it a directed multi-graph), because we may generate multiple candidate programs between two patterns (some of which incorrect), and furthermore certain patterns can be inherently ambiguous with multiple meanings (e.g., `mm/dd/yyyy` vs. `dd/mm/yyyy`). We omit such edges in the example graph for simplicity, but our techniques below still apply when multiple edges are present.

5.3 Harvest TBP programs by relationships

Given a global TBP graph, in this section we discuss an automated approach to harvest TBP programs leveraging relationships between programs. We will discuss an alternative that leverages human labels in Section 5.4.

From a global TBP graph, we exploit two types of implicit relationships between TBP programs in order to infer their quality. Intuitively, this can be seen as asking programs to "corroborate" each other's validity.

Lossless inverse programs. The first type of relationship is referred to as *inverse programs*, defined as follows.

Definition 4. Two TBP programs (P, P', T) and (P', P, T') are *lossless inverse programs*, if applying T on column C matching P (or $P \in \mathbf{P}(C)$) produces $T(C)$ of pattern P', from which applying T' produces the original input C, or $T'(T(C)) = C$.

Inverse programs are similar in spirit to inverse-functions in mathematics, and we write such pairs as (P, P', T) and (P', P, T^{-1}). Here, if after applying T and T' sequentially we obtain the original input data C, it is a good indication that (1) both T and T' are *lossless* transformations (for otherwise one could not regenerate an identical C); and (2) both T and T' are likely good TBP programs (because the counterpart is generated independently).

Example 7. In the example graph in Figure 9, we can see many pairs of inverse programs, such as the pair (P_8, P_9, T_{89}) and (P_9, P_8, T_{98}) for geo-coordinates; the pair (P_4, P_5, T_{45}) and (P_5, P_4, T_{45}) for date-time strings, etc.

It can be verified that such transformations are lossless, because executing T and T^{-1} produces output identical to the input. For instance, from P_5 to P_4 and applying T_{54}, even though some tokens are dropped (e.g., "`Sun` and "`Dec`"), it is still lossless because when we apply T_{45} in the other direction, we re-produce the original input.

While the example graph in Figure 7 would suggest that inverse programs can be identified as cycles of length 2, in general we need to test whether the inverse-relationship of Definition 4 holds on T and T' using real data.

Specifically, in order to test if (P, P', T) and (P', P, T') are inverse-programs, we leverage column-pairs (C, C') from $\mathbf{T}_{CCT}$ for which (P, P', T) is known to hold. We then apply (P', P, T') in the other direction, by applying T' on each C' to test whether the original C can be re-produced.

For each candidate pair TBP (P, P', T) and (P', P, T'), we perform this test on $\mathbf{T}_{CCT}$, and compute a success rate for inversion, denoted by $S_{\text{inv}}((P, P', T), (P', P, T'))$:

$$S_{\text{inv}} = \frac{|\{(C, C', T) \in \mathbf{T}_{CCT}, P \in \mathbf{P}(C), P' \in \mathbf{P}(C'), T'(T(C)) = C\}|}{|\{(C, C', T) \in \mathbf{T}_{CCT}, P \in \mathbf{P}(C), P' \in \mathbf{P}(C'), T(C) = C'\}|} \quad (3)$$

We consider (P, P', T) and (P', P, T') be inverse programs if the inverse relationship test holds on a large fraction of real data tested (e.g., $S_{\text{inv}} > 0.8$). In such cases, we can consider both as quality TBP programs, because they are

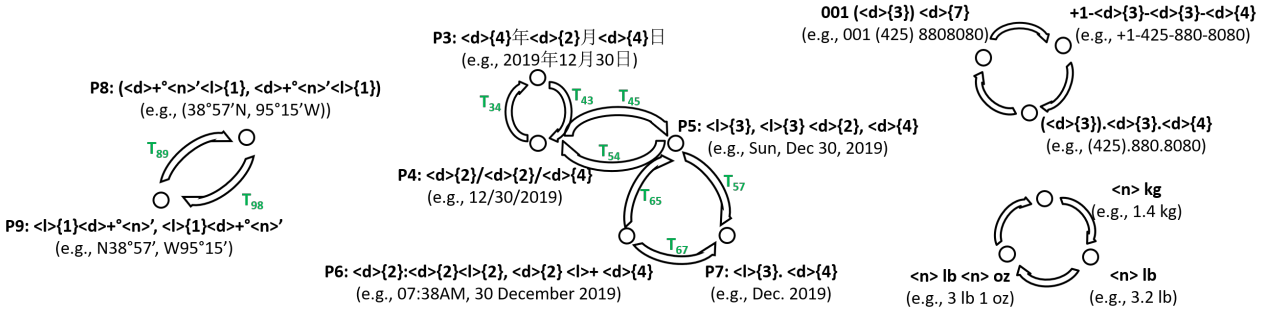


Figure 9: An example TBP graph (with some edges omitted to avoid clutter). Each vertex corresponds to a data pattern P, and each edge (P, P', T) is a program between two patterns P and P'. (We use short-hand notation of pattern tokens, such as `<d>`, `<l>` and `<n>` that stand for `<digit>`, `<letter>` and `<num>`, respectively).

lossless and can corroborate each other. Spurious programs "over-fitted" on limited examples in the TBE step would often fail the test.

Triangular equivalent programs. Because not all transformations are lossless (a prerequisite for inverse-programs), we also consider a second type of *triangular* relationship between programs.

Definition 5. Three programs (P, P'', T), (P, P', T') and (P', P'', T'') are defined as *triangular equivalent programs*, if applying T on column C matching P (or $P \in \mathbf{P}(C)$) produces output $T(C)$, which is identical to applying T' followed by T'' sequentially on C, or $T''(T'(C)) = T(C)$.

Example 8. Consider the triangle between P_5, P_6 and P_7 in Figure 9. It can be seen that the program T_{67} is lossy (the time part is dropped after the transformation), and thus cannot be part of an inverse program.

However, applying T_{67} on suitable input (e.g. `07:38AM, 30 December 2019`) produces output (`Dec. 2019`) that is identical to applying T_{65} followed by T_{57}, suggesting a triangular relationship, which can be used as a piece of evidence to substantiate the validity of T_{65}.

Like inverse-programs, we test each triple (P, P'', T), (P, P', T') and (P', P'', T''), by performing tests on column data in $\mathbf{T}_{CCT}$. The success rate S_{tri} can be calculated as:

$$\frac{|\{(C, C'', T) \in \mathbf{T}_{CCT}, P \in \mathbf{P}(C), P'' \in \mathbf{P}(C''), T''(T'(C)) = C''\}|}{|\{(C, C'', T) \in \mathbf{T}_{CCT}, P \in \mathbf{P}(C), P'' \in \mathbf{P}(C''), T(C) = C''\}|} \tag{4}$$

We consider triangular-equivalence to hold on a program-triple, if the test above holds on most column pairs from $\mathbf{T}_{CCT}$ (e.g., $S_{\text{tri}} > 0.8$).

Harvest TBP programs by program relationships. We note that because the program relationships above can identify high-quality TBP programs, this provides an automated approach to harvest TBP programs. We implement tests of inverse and triangular relationships as Map-Reduce style jobs, using success-rates defined in Equation (3) and Equation (4).

5.4 Harvest TBP programs by curation

We note that there are many application scenarios where suggested TBP-transformations are required to be close to 100% correct (e.g., suggesting data-repairs in Excel or Google Sheets). Such TBP programs need to be manually inspected and verified beforehand.

As a result, we also consider a problem variant where TBP programs have to be verified by human curators, who can inspect and verify up to k candidate programs, and label them as correct or incorrect. The key technical challenge is to select programs of high "impact" for humans to verify, so that the benefit of the k labels can be maximized.

For this task, we start with the graph where edges/programs are already verified as inverse or triangular. Recall that each edge/program has a "coverage" score $\text{Cov}(P, P', T)$, indicating the popularity/importance of the program. Intuitively, frequently-used transformations (e.g., for common date-time formats) have high coverage scores and are more important to be verified first. Our overall objective is thus to maximize total coverage scores of edges/programs given a budget of k labels.

Our observation here is that because of relationships between programs, verifying a program on one edge can have *super-modular* benefits, as shown in the example below.

Example 9. In the curation setting, each edge/program in Figure 9 needs to be verified and has an associated coverage score. Observe that if a human curator can verify T_{89} to be correct, then the inverse T_{98} is verified implicitly and be assumed correct. We thus "gain" the coverage-scores on both T_{89} and T_{98} by verifying one edge. Similarly, if both T_{65} and T_{57} are verified as correct, T_{67} is also likely correct (because of the triangular relationship), allowing us to obtain the coverage score on T_{67} without labeling it.

Given that we try to maximize total coverage scores, and the inverse/triangular relationships that we can leverage (e.g., verifying an edge also implicitly verifies the inverse of the edge), the incremental benefit of labeling an edge is "super-modular" in regions of the graph with a dense cluster of edges (e.g., the middle part of Figure 9). In comparison, verifying an edge not well-connected to other nodes would have reduced impact.

We formulate this as an optimization problem. Given a TBP graph $G = (V, E)$, where each edge $e \in E$ has a coverage score $\text{Cov}(e)$. Our objective is to find a subset of edges $E_s \subset E$ to verify, with $|E_s| \leq k$, such that the total coverage score of these verified programs, together with ones implicitly verified through program relationships, is maximized.

We write this in an ILP formulation, termed as CMPS (coverage-maximizing program selection) below:

$$\text{(CMPS)} \max \sum_{e_i \in E} \text{Cov}(e_i) v_i \tag{5}$$

$$\text{s.t.} \sum_{e_i \in E} x_i \leq k \tag{6}$$

$$y_m \leq x_i + x_j, \quad \forall\, \text{Inv}_m(e_i, e_j) \in \text{Inv}(G) \tag{7}$$

$$z_n \leq x_i + x_j + x_l - 1, \forall \text{Tri}_n(e_i, e_j, e_l) \in \text{Tri}(G) \tag{8}$$

$$v_i \leq x_i + \sum_{e_i \in \text{Inv}_m} y_m + \sum_{e_i \in \text{Tri}_n} z_n, \quad \forall e_i \in E \tag{9}$$

$$v_i, x_i, y_m, z_n \in \{0, 1\} \tag{10}$$

Here, x_i indicates whether e_i is selected for human verification, y_m indicates whether the m-th inverse-relationship, denoted by Inv_m, has a participating edge selected for verification, z_n indicates whether n-th triangular-equivalence relationship, Tri_n, have more than two edges selected for verification. And finally v_i indicates whether e_i can be treated as correct, through explicit human verification, or program relationships. All of these are $\{0, 1\}$ binary variables.

The objective function in Equation (5) calculates the sum of coverage scores of all programs implicitly or explicitly verified (indicated by v_i). Equation (6) ensures that at most k edges are explicitly verified by humans. Equation (7) and Equation (8) check whether enough edges in each inverse/-triangular relationship are explicitly verified by humans (and if so the corresponding y_m and z_n is set to 1). Finally, Equation (9) checks whether e_i can be verified explicitly (through x_i) or implicitly (through y_m or z_n).

It is worth noting that this problem is *super-modular* [46], because adding one labeled edge brings more benefit on a larger set of selected edges S than a smaller set S'. Unlike *sub-modular* maximization problems (e.g., Max-Coverage [15]) super-modular maximization is more difficult. We show the hardness of CMPS below.

THEOREM 1. The CMPS problem is NP-hard, and no PTAS likely exist under standard complexity assumptions.

A proof of this result can be obtained using a reduction from densest k-subgraph [43].

Given the hardness of CMPS, in the curation setting, we resort to a heuristic that at each step, picks the edge with maximum benefit (via explicit and implicit verification), until the budget k is exhausted. We omit pseudo-code of this procedure in the interest of space.

Additional formulations. There are a few alternative formulations to the curation problem, which are variants of CMPS. One possible alternative models the "coverage" $Cov(e_i)$ of each edge/program e as the set of column-pairs this program covers (as opposed to using a numeric count like in CMPS). This would take into account possible overlap in coverage between edges/programs, and can more accurately model the benefit of verifying an edge.

A second alternative formulation additionally models the likelihood of an edge being correct, using estimated accuracy scores, which generalizes CMPS in a different dimension.

From a complexity perspective, because CMPS is a special case of these formulations, these formulations are at least as hard as CMPS. We defer details of these formulations a full version of the paper.

6. EXPERIMENTS

In our evaluations, we set out to answer two main questions: (1) whether the proposed method can harvest useful TBP programs in an unsupervised manner, and (2) whether the curated variant can leverage human labels effectively.

6.1 Datasets and Evaluation

In our experiments, we use a recent crawl of web tables extracted from the index of a search engine [17], and prepare five different table corpora that are summarized in Table 3.

• **WEB**. WEB-en contains a set 135M relational tables in English extracted from the web [17].

• **WIKI**. In order to test whether AUTO-TRANSFORM can "generalize" to different data corpora (e.g., in different languages or domains), we also perform tests using Wikipedia tables in 4 popular languages: English (WIKI-en), Spanish (WIKI-es), Japanese (WIKI-ja), Chinese (WIKI-zh).[11] Specifically, we learn TBP programs using Wikipedia tables in these 4 languages, and then test such programs to Auto-Repair 1% of Wiki-en, 10% of Wiki-es, 10% of Wiki-ja and 10% of Wiki-zh tables, respectively (shown in Table 4). We also evaluate the quality of the programs so produced on each corpus.

Table 3: Characteristics of table corpus used.

	language	# of tables	# of columns	# of linked col-pairs
WEB-en	English	135M	350M	1.2B
WIKI-en	English	7.3M	41M	232M
WIKI-es	Spanish	1.1M	6.3M	13M
WIKI-ja	Japanese	790K	4.7M	5M
WIKI-zh	Chinese	1.8M	9M	10M

Table 4: Four disjoint test corpora in different languages.

Corpus to learn TBP	Corpus to test Auto-Repair
WEB-en	1% WIKI-en
WIKI-es/en	10% WIKI-es
WIKI-ja/en	10% WIKI-ja
WIKI-zh/en	10% WIKI-zh

There are non-trivial scalability challenges given the large number of tables. We carefully engineer the pipelines outlined in Figure 6 as multiple map-reduce-style jobs executed on a production cluster [16]. These jobs run in under 10 hours end-to-end.

Since each TBP program (and repairs it produces) has a confidence score (Equation (3) and (4)), we rank programs (resp. repairs) by confidence, and manually label top-K programs (resp. repairs). Following a standard from the IR literature [29, 57], we use a ternary labeling, where in addition to results that are clearly correct or incorrect, we also explicitly label suggested transformations that are reasonable but not always desired (e.g., in a column with mixed year values like `1997` and `1998`*, recommend a transformation that removes all * symbols). For all methods compared, results labeled as "reasonable" would not count in the recall, but also do not count against the precision.

We report quality results of top-K using the *precision@K* metric [49], defined as $\frac{\text{\#-correct-repairs-top-K}}{\text{\#-correct-top-K+\#-incorrect-top-K}}$, and $\frac{\text{\#-correct-programs-top-K}}{\text{\#-correct-top-K+\#-incorrect-top-K}}$, respectively.

6.2 Methods Compared

We compare the following methods for Auto-Repair, which is one application of TBP discussed in the introduction.

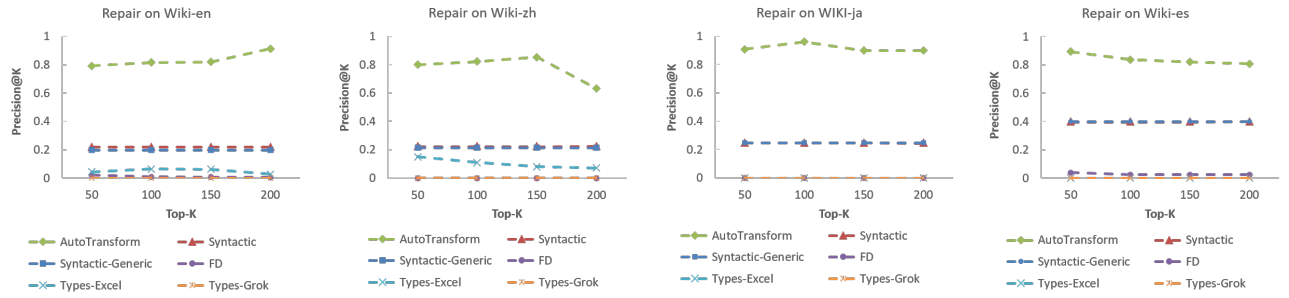
• **Auto-Transform**. This uses TBP to repair values. We apply a TBP program (P, P', T) on a column C, when there are two values $v, v' \in C$ whose patterns match P, P', respectively. We suggest $T(v)$ in place of v in such cases.

• **Syntactic**. System-A[12] is a commercial data preparation tool with a pattern-based standardization feature that is the closest to TBP.[13] We refer to this as Syntactic, since based on public documentation it uses two groups of curated syntactic-rules: (1) "Generic Conversion" defines a set of permitted transformations between source/target patterns. For example, symbol changes are allowed (e.g., from

[11] While experiments were conducted on table data across 12 languages as listed in Section 3.2, the authors are only familiar with these 4 languages, which are chosen for labeling.

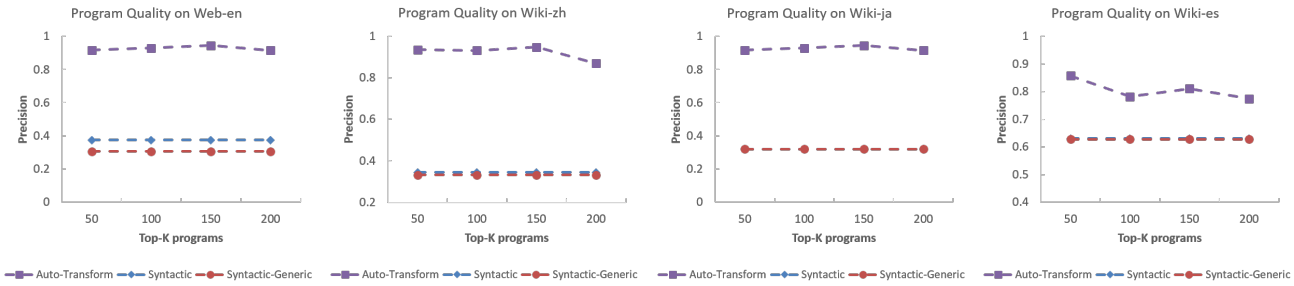
[12] We anonymize the name of the system in compliance with its EULA (also a tradition in database benchmarking).

[13] Because System-A is a UI-driven tool with no API exposed for large-scale testing, we implemented it based on public documentations of this feature on its website.



(a) Wiki-en (b) Wiki-zh (c) Wiki-ja (d) Wiki-es

Figure 10: Quality of repairs on Wiki-en, Wiki-zh, Wiki-ja, Wiki-es, using TBP programs learned from corresponding corpus.



(a) Web-en (b) Wiki-zh (c) Wiki-ja (d) Wiki-es

Figure 11: Quality of TBP programs produced on Web-en, Wiki-zh, Wiki-ja, Wiki-es, respectively.

"`123.456.7890`" to "`123-456-7890`"), so is splitting (e.g., from "`1234567890`" to "`123-456-7890`"), etc. (2) "Datetime Conversion": if a column is recognized as datetime (based on known date-time formats), then datetime-specific rules kick in, which allow transformations on up to two token-groups (e.g., from "`Jan 1, 1981`" to "`01/01/1981`"), but not more than two (e.g., from "`Jan 1, 1981`" to "`01/01/81`").

While these curated rules provide a strong baseline that handles many common transformations, they can also produce false-positives that are not entirely intuitive to humans – for example, in a column with `<alpha>+ (<digit>.<digit>)` (e.g. "`Washington (8.5)`") and `<alpha>+ (<digit>{2})` (e.g., "`Washington (12)`"), a suggestion to transform "`Washington (8.5)`" into "`Washington (85)`" will be produced. This is because dropping the symbol "`.`" is allowed by rule, even though it is not semantically meaningful.

• **Syntactic-Generic**. This is Syntactic but with only generic conversion rules as defined in System-A documentation. We ran both Syntactic and Syntactic-Generic on the entire test corpus, and label their respective top-K results.

• **Grok-Types**. An alternative to repairing data format issues in columns, is to use predefined regex patterns to detect known data-types (e.g., date-time, email, url, ip, etc.). If more than one known pattern/format is detected in the same column (e.g., date-format-1 and date-format-2), it is likely a format issue that needs to be fixed.

We use Grok-patterns [3] for type-detection, which has over 70 curated regex patterns for common data-types. For each repair from Auto-Transform, we evaluate if the same issue can be detected by Grok (by testing if there are two Grok patterns in the column), and if so we mark it as "fixable" by Grok (even though no repair exists in Grok). We report the total number of fixable issues for Grok.

• **Excel-Types**. Observing that Excel can auto-format data of certain known types (e.g., date-time and currency) into standard formats [1], we simulate the Excel logic from [5] (which has over 110 date-time formats for "`en-us`" alone). Like in Grok-Types, this is to understand the potential coverage of Excel types – specifically, for each real issue detected by Auto-Transform, we check if Excel type-detection logic can discover two known formats mixed in the same column. We again report the fraction of issues fixed by Auto-Transform that are also fixable by Excel.

• **Functional Dependency (FD)**. First-order logic like FD is widely used for error detection and repair (e.g., [11, 13, 23]), which conceptually addresses an orthogonal type of errors manifested as inconsistency across multiple columns (whereas TBP leverage information within single columns).

Even though the errors addressed by the two are largely complementary, we nevertheless perform a comparison to quantify the possible overlap between FD-based approaches and TBP-based methods. Specifically, if the column C of tuple t (written as $C(t)$) is fixed by Auto-Transform from v to $\bar{v}$, we check whether there is *any* approximate FD: $C_l \rightarrow C$ in the same table (with column C_l being the LHS), that can possibly fix v to $\bar{v}$. Namely, we check if there exists another tuple t' with $C_l(t') = C_l(t)$ and $C(t') = \bar{v}$, which would make the fix $v \rightarrow \bar{v}$ *possible*. We report the fraction of issues fixed by Auto-Transform that are fixable by FD.

6.3 Experiment Results

TBP-based Data Repairs. Figure 10 shows the quality of top-K repairs generated on Web-en, Wiki-zh, Wiki-ja, Wiki-es, respectively, using TBP programs generated from corresponding corpora listed in Table 4.

Overall the trend is consistent across the 4 test corpora. Auto-Transform can generate high-quality repairs across different languages (examples of which are shown in Figure 3). Note that these TBP programs are harvested using the graph-based analysis without human curation (Section 5.3). This experiment shows that our approach can indeed generalize across different types of table corpora (in

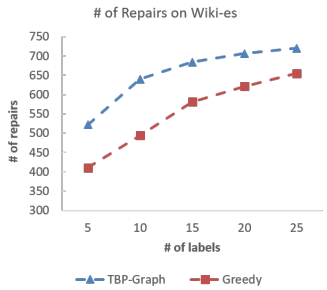


Figure 12: Benefit of providing k labels to curate programs using TBP-Graph and Greedy, measured as the number of data repairs possible on 10% of Wiki-es.

this case different languages), and provide a way to repair data errors complementary to existing methods.

Also note that although we only label top-200 auto-repair predictions, because these top-200 are produced on small samples of each test corpus (e.g., 1% sample of Wiki-en), we can extrapolate that this corresponds a large number of good repairs on the full test corpus.

We analyze the false-positives in our results, and find that number-formatting to be a main cause. For example, we were able to identify as training data, paired-columns with identical numbers in different formats (e.g., "`$12345`" vs. "`$12,345`). On such columns, a simple inferred TBP would produce "`<digit>+`" and "`<digit>+,<digit>+`" as input/output patterns, with a transformation that inserts one comma after the third character counting from the back. This however yields false-positives on longer input (e.g., "`$1234567`"). Extending the current pattern language with more expressive constructs (e.g., variable-length groups) would be useful extensions.

Syntactic produces the second-best result, repairing up to 40% suggested results correctly[14]. We notice a significant number of false-positives, like the case of suggesting to fix "`Washington (8.5)`" to "`Washington (85)`" discussed above (such a transformation of dropping punctuation is allowed by static rules, but is not semantically meaningful). Syntactic-Generic produces results that are slightly worse than Syntactic, because curated date-time repairs are typically correct.

When we analyze how many of the issues repaired by Auto-Transform are fixable by Grok-Types and Excel-Types, we observe a very small overlap (up to around 20%), despite the fact that these two approaches use large numbers of predefined format patterns. The difference is more pronounced on Wiki-ja and Wiki-es, where the coverage of both Grok-Types and Excel-Types are close to 0 (for these comparisons we configure language/locale of Excel-Types to Japanese and Spanish, respectively). This comparison suggests that the types of issues addressed by Auto-Transform are likely not fixable using a curated list of data-type format strings.

We perform a similar coverage analysis for FD – for each issue identified by Auto-Transform, we programmatically test whether it is a possible fix using approximate FD. We find minimal overlap between FD and Auto-Transform across the 4 test sets. This suggests that at least for web-tables (which tend to be small), there is little data redundancy across multiple columns that FD can leverage to suggest repairs (TBP in comparison uses only values in a single column). We believe this suggests that Auto-Transform can be an interesting alternative that can fix data issues complementary to existing FD-base approaches.

TBP Program Quality. Figure 11 shows the quality of top-K TBP programs generated across 4 corpora. We perform a similar quality evaluation using precision@K, by labeling TBP programs learned on different corpus. We compare with the repair programs generated by Syntactic and Syntactic-Generic on these corpora. Because Syntactic and Syntactic-Generic have no inherent notion of confidence, we randomly sample 200 programs and evaluate precision, which translates to two flat lines in the figure.

Like we discussed above, Auto-Transform substantially outperforms these rule-based methods, because not all syntactic programs they generate are semantically correct.

Impact of Curation. Figure 12 shows an experiment where humans can manually label k edges/programs as correct or incorrect. We refer to our CMPS approach as TBP-Graph (Section 5.4), and compare with Greedy, which is a heuristic that prioritizes labeling based on individual edges.

We evaluate the benefit of curating top-K programs, measured as the total number of auto-repairs that these curated programs can perform on tables in Web-es. We observe in Figure 12 that, TBP-Graph is superior over Greedy, because it considers the super-modular benefit between programs/edges (similar results are observed on other corpus). It is encouraging to see that a substantial number of issues in Wikipedia tables can be repaired with a handful of labels.

7. RELATED WORKS

Data transformation. Data transformation is a long-standing problem in the literature [4, 14, 26, 27, 50, 53, 58]. A notable line of work is "transform-by-example" (TBE) [10, 31, 33, 35, 36, 41, 42, 59], in which users provide a few (typically two or three) paired input/output examples to specify an intent, and the system would search for programs consistent with all given examples. TBE technologies have generated substantial impacts on commercial software, and are now available as features in popular systems such as Microsoft Excel [31], Power BI [33], and Trifacta [6]. Variants of TBE have also been developed for scenarios where paired-examples can be inferred [25, 68].

Data error detection and repair. Detecting and repairing data quality issues is also an important topic [9, 38]. The community has produced a plethora of methods for error-detection [12, 19, 20, 24, 30, 37, 39, 40, 45, 48, 51, 52, 60, 65]), as well as many related methods to repair them [11, 13, 23, 21, 22, 55, 61, 63]. The TBP-based data repair provides an alternative that is largely complementary to existing constraint-based methods.

8. CONCLUSIONS

We propose transform-by-pattern (TBP), a new approach to data transformation based on input/output data patterns only, which enables applications such as suggesting transformations for data repairs. We believe exploring new ways to describe input/output columns beyond syntactic regex patterns can be an interesting direction for future work.

[14]Note that it produces a flat line, because the static rules it uses have no inherent notion of confidence and we label a sample of 200 (un-ordered) repair predictions from Syntactic.

9. REFERENCES

[1] changing numbers to dates. https://aka.ms/stop-excel-change-date-formats.

[2] Dresden web tables corpus. https://wwwdb.inf.tu-dresden.de/misc/dwtc/.

[3] Grok data patterns in elasticsearch (retrieved 2019-11. https://github.com/elastic/elasticsearch/blob/master/libs/grok/src/main/resources/patterns/grok-patterns.

[4] Informatica Advanced Data Transformation. https://www.informatica.com/products/data-integration/advanced-data-transformation.html.

[5] Oadate in excel (retrieved 2019-11. https://docs.microsoft.com/en-us/dotnet/api/system.datetime.fromoadate?redirectedfrom=MSDN&view=netframework-4.8#System_DateTime_FromOADate_System_Double_, https://docs.microsoft.com/en-us/dotnet/api/system.globalization.datetimeformatinfo.getalldatetimepatterns?view=netframework-4.8.

[6] Transform-by-Example feature in Trifacta. https://www.trifacta.com/blog/transform-by-example-your-data-cleaning-wish-is-our-command.

[7] Trifacta: Standardize using patterns (retrieved 2019-03). https://docs.trifacta.com/display/SS/Standardize+Using+Patterns#StandardizeUsingPatterns-PatternsbyExample.

[8] Web data commons - web tables corpus. http://km.aifb.kit.edu/sites/webdatacommons/webtables/index.html.

[9] Z. Abedjan, X. Chu, D. Deng, R. C. Fernandez, I. F. Ilyas, M. Ouzzani, P. Papotti, M. Stonebraker, and N. Tang. Detecting data errors: Where are we and what needs to be done? *VLDB*, 9(12), 2016.

[10] Z. Abedjan, J. Morcos, I. F. Ilyas, M. Ouzzani, P. Papotti, and M. Stonebraker. Dataxformer: A robust transformation discovery system. In *ICDE*, 2016.

[11] F. N. Afrati and P. G. Kolaitis. Repair checking in inconsistent databases: algorithms and complexity. In *Proceedings of the 12th International Conference on Database Theory*. ACM, 2009.

[12] L. Berti-Equille, H. Harmouch, F. Naumann, N. Novelli, and T. Saravanan. Discovery of genuine functional dependencies from relational data with missing values [abstract for inforsid 2019]. In *INFORSID 2019*, 2019.

[13] P. Bohannon, W. Fan, M. Flaster, and R. Rastogi. A cost-based model and effective heuristic for repairing constraints by value modification. In *SIGMOD*, 2005.

[14] S. Bowers and B. Ludäscher. An ontology-driven framework for data transformation in scientific workflows. In *International Workshop on Data Integration in the Life Sciences*, pages 1–16. Springer, 2004.

[15] N. Buchbinder, M. Feldman, J. Naor, and R. Schwartz. Submodular maximization with cardinality constraints. In *Proceedings of the twenty-fifth annual ACM-SIAM symposium on Discrete algorithms*, pages 1433–1452. SIAM, 2014.

[16] R. Chaiken, B. Jenkins, P.-Å. Larson, B. Ramsey, D. Shakib, S. Weaver, and J. Zhou. Scope: easy and efficient parallel processing of massive data sets. *Proceedings of the VLDB Endowment*, 1(2):1265–1276, 2008.

[17] K. Chakrabarti, S. Chaudhuri, Z. Chen, K. Ganjam, and Y. He. Data services leveraging bing's data assets. *IEEE Data Eng. Bull.*, 2016.

[18] K. Chakrabarti, S. Chaudhuri, Z. Chen, K. Ganjam, Y. He, and W. Redmond. Data services leveraging bing's data assets. *IEEE Data Eng. Bull.*, 39(3):15–28, 2016.

[19] F. Chiang and R. J. Miller. Discovering data quality rules. *Proceedings of the VLDB Endowment*, 1(1):1166–1177, 2008.

[20] X. Chu, I. F. Ilyas, and P. Papotti. Discovering denial constraints. *Proceedings of the VLDB Endowment*, 6(13):1498–1509, 2013.

[21] X. Chu, I. F. Ilyas, and P. Papotti. Holistic data cleaning: Putting violations into context. In *ICDE*. IEEE, 2013.

[22] 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.

[23] G. Cong, W. Fan, F. Geerts, X. Jia, and S. Ma. Improving data quality: Consistency and accuracy. In *VLDB*, 2007.

[24] T. Dasu, T. Johnson, S. Muthukrishnan, and V. Shkapenyuk. Mining database structure; or, how to build a data quality browser. In *SIGMOD*, 2002.

[25] D. Deng, W. Tao, Z. Abedjan, A. Elmagarmid, I. F. Ilyas, G. Li, S. Madden, M. Ouzzani, M. Stonebraker, and N. Tang. Unsupervised string transformation learning for entity consolidation. In *2019 IEEE 35th International Conference on Data Engineering (ICDE)*, pages 196–207. IEEE, 2019.

[26] S. Dessloch, M. A. Hernández, R. Wisnesky, A. Radwan, and J. Zhou. Orchid: Integrating schema mapping and etl. In *2008 IEEE 24th International Conference on Data Engineering*, pages 1307–1316. IEEE, 2008.

[27] R. Dhamankar, Y. Lee, A. Doan, A. Halevy, and P. Domingos. imap: Discovering complex semantic matches between database schemas. In *SIGMOD*, SIGMOD '04, pages 383–394, New York, NY, USA, 2004. ACM.

[28] K. Fisher, D. Walker, K. Q. Zhu, and P. White. From dirt to shovels: fully automatic tool generation from ad hoc data. *ACM SIGPLAN Notices*, 43(1):421–434, 2008.

[29] C. W. Gleverdon and C. W. Cleverdon. Report on the testing and analysis of an investigation into the comparative efficiency of indexing systems. 1962.

[30] L. Golab, H. Karloff, F. Korn, and D. Srivastava. Data auditor: Exploring data quality and semantics using pattern tableaux. *VLDB*, 3(1-2), 2010.

[31] S. Gulwani. Automating string processing in spreadsheets using input-output examples. In *ACM Sigplan Notices*, volume 46, pages 317–330. ACM, 2011.

[32] J. Hare, C. Adams, A. Woodward, and H. Swinehart. Forecast snapshot: Self-service data preparation, worldwide, 2016. *Gartner, Inc.*, February 2016.

[33] Y. He, X. Chu, K. Ganjam, Y. Zheng, V. Narasayya, and S. Chaudhuri. Transform-data-by-example (tde): an extensible search engine for data transformations. *Proceedings of the VLDB Endowment*, 11(10):1165–1177, 2018.

[34] Y. He, K. Ganjam, and X. Chu. Sema-join: joining semantically-related tables using big table corpora. *Proceedings of the VLDB Endowment*, 8(12):1358–1369, 2015.

[35] Y. He, K. Ganjam, K. Lee, Y. Wang, V. Narasayya, S. Chaudhuri, X. Chu, and Y. Zheng. Transform-data-by-example (tde) extensible data transformation in excel. In *Proceedings of the 2018 International Conference on Management of Data*, pages 1785–1788, 2018.

[36] J. Heer, J. M. Hellerstein, and S. Kandel. Predictive interaction for data transformation. In *CIDR*, 2015.

[37] A. Heidari, J. McGrath, I. F. Ilyas, and T. Rekatsinas. Holodetect: Few-shot learning for error detection. In *Proceedings of the 2019 International Conference on Management of Data*, pages 829–846, 2019.

[38] J. M. Hellerstein. Quantitative data cleaning for large databases. *United Nations Economic Commission for Europe (UNECE)*, 2008.

[39] Z. Huang and Y. He. Auto-detect: Data-driven error detection in tables. In *Proceedings of the 2018 International Conference on Management of Data*, pages 1377–1392, 2018.

[40] I. F. Ilyas, V. Markl, P. Haas, P. Brown, and A. Aboulnaga. Cords: automatic discovery of correlations

and soft functional dependencies. In *SIGMOD*, 2004.

[41] Z. Jin, M. R. Anderson, M. Cafarella, and H. V. Jagadish. Foofah: Transforming data by example. In *SIGMOD*, 2017.

[42] Z. Jin, M. Cafarella, H. Jagadish, S. Kandel, M. Minar, and J. M. Hellerstein. Clx: Towards verifiable pbe data transformation. *arXiv preprint arXiv:1803.00701*, 2018.

[43] S. Khot. Ruling out ptas for graph min-bisection, dense k-subgraph, and bipartite clique. *SIAM Journal on Computing*, 36(4):1025–1071, 2006.

[44] R. Kimball and J. Caserta. *The data warehouse ETL toolkit: practical techniques for extracting, cleaning, conforming, and delivering data*. John Wiley & Sons, 2011.

[45] J. Kivinen and H. Mannila. Approximate inference of functional dependencies from relations. *Theoretical Computer Science*, 149(1), 1995.

[46] B. Korte, J. Vygen, B. Korte, and J. Vygen. *Combinatorial optimization*, volume 2. Springer, 2012.

[47] Y. Li, J. Li, Y. Suhara, A. Doan, and W.-C. Tan. Deep entity matching with pre-trained language models. *arXiv preprint arXiv:2004.00584*, 2020.

[48] M. Mahdavi, Z. Abedjan, R. Castro Fernandez, S. Madden, M. Ouzzani, M. Stonebraker, and N. Tang. Raha: A configuration-free error detection system. In *Proceedings of the 2019 International Conference on Management of Data*, pages 865–882, 2019.

[49] C. D. Manning, P. Raghavan, and H. Schütze. *Introduction to information retrieval*. Cambridge university press, 2008.

[50] P. McBrien and A. Poulovassilis. Data integration by bi-directional schema transformation rules. In *Proceedings 19th International Conference on Data Engineering (Cat. No. 03CH37405)*, pages 227–238. IEEE, 2003.

[51] F. Neutatz, M. Mahdavi, and Z. Abedjan. Ed2: A case for active learning in error detection. In *Proceedings of the 28th ACM International Conference on Information and Knowledge Management*, pages 2249–2252, 2019.

[52] A. Qahtan, N. Tang, M. Ouzzani, Y. Cao, and M. Stonebraker. Anmat: automatic knowledge discovery and error detection through pattern functional dependencies. In *Proceedings of the 2019 International Conference on Management of Data*, pages 1977–1980, 2019.

[53] E. Rahm and H. H. Do. Data cleaning: Problems and current approaches. *IEEE Data Eng. Bull.*, 23(4):3–13, 2000.

[54] V. Raman and J. M. Hellerstein. Potter's wheel: An interactive data cleaning system. In *VLDB*, volume 1, 2001.

[55] T. Rekatsinas, X. Chu, I. F. Ilyas, and C. Ré. Holoclean: Holistic data repairs with probabilistic inference. *VLDB*, 10(11), 2017.

[56] R. L. Sallam, P. Forry, E. Zaidi, and S. Vashisth. Gartner: Market guide for self-service data preparation. 2016.

[57] M. Sanderson. *Test collection based evaluation of information retrieval systems*. Now Publishers Inc, 2010.

[58] A. Simitsis, P. Vassiliadis, and T. Sellis. Optimizing etl processes in data warehouses. In *21st International Conference on Data Engineering (ICDE'05)*, pages 564–575. IEEE, 2005.

[59] R. Singh. Blinkfill: Semi-supervised programming by example for syntactic string transformations. *Proceedings of the VLDB Endowment*, 9(10):816–827, 2016.

[60] P. Wang and Y. He. Uni-detect: A unified approach to automated error detection in tables. In *Proceedings of the 2019 International Conference on Management of Data*, pages 811–828, 2019.

[61] Y. Wang and Y. He. Synthesizing mapping relationships using table corpus. In *Proceedings of the 2017 ACM International Conference on Management of Data*, pages 1117–1132, 2017.

[62] R. Wu, S. Chaba, S. Sawlani, X. Chu, and S. Thirumuruganathan. Zeroer: Entity resolution using zero labeled examples. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*, pages 1149–1164, 2020.

[63] M. Yakout, L. Berti-Équille, and A. K. Elmagarmid. Don't be scared: use scalable automatic repairing with maximal likelihood and bounded changes. In *SIGMOD*, 2013.

[64] C. Yan and Y. He. Synthesizing type-detection logic for rich semantic data types using open-source code. In *Proceedings of the 2018 International Conference on Management of Data*, pages 35–50, 2018.

[65] J. N. Yan, O. Schulte, J. Wang, and R. Cheng. Coded: Column-oriented data error detection with statistical constraints.

[66] D. Zhang, Y. Suhara, J. Li, M. Hulsebos, Ç. Demiralp, and W.-C. Tan. Sato: Contextual semantic type detection in tables. *arXiv preprint arXiv:1911.06311*, 2019.

[67] C. Zhao and Y. He. Auto-em: End-to-end fuzzy entity-matching using pre-trained deep models and transfer learning. In *The World Wide Web Conference*, pages 2413–2424, 2019.

[68] E. Zhu, Y. He, and S. Chaudhuri. Auto-join: Joining tables by leveraging transformations. *Proceedings of the VLDB Endowment*, 10(10):1034–1045, 2017.