Foofah: A Programming-By-Example System for Synthesizing Data Transformation Programs

Zhongjun Jin Michael R. Anderson Michael Cafarella H. V. Jagadish
University of Michigan, Ann Arbor
{markjin,mrander,michjc,jag}@umich.edu

ABSTRACT

Advancements in new data analysis and visualization technologies have resulted in wide applicability of data-driven decision making. However, raw data from various sources must be wrangled into a suitable form before they are processed by the downstream data tools. People traditionally write data transformation programs to automate this process, and such work is cumbersome and tedious.

We built a system called FOOFAH for helping the user easily synthesize a desired data transformation program. Our system minimizes the user's effort by only asking for a small illustrative example comprised of the raw input data and the target transformed output; FOOFAH then synthesizes a program that can perform the desired data transformation. This demonstration showcases how the user can apply FOOFAH to real-world data transformation tasks.

Keywords

Data Transformation; Program Synthesis; Programming By Example; A* algorithm; Heuristic

1. PROBLEM AND MOTIVATION

Data transformation is a crucial first step that reorganizes raw data (Table 1 for example) into a format that can be easily consumed by databases or data analytics tools (Table 2). Manually wrangling the raw data using tools like Excel is inefficient and cumbersome when the size of the data is large. Traditionally, programmers write data-specific scripts to automate the transformation process. However, the job of handwriting data transformation programs is still tedious and labor-intensive: these programs are tailored to particular data sources and are not able to adapt newly acquired data sources. Further, writing a correct program is often hard. In practice, analysts normally spend more time preparing data than analyzing it; up to 80% of a data scientist's time can be spent on transforming data into a usable state [10].

Recent research into automated and assisted data transformation systems has focused on reducing the user's time and required skill level, with some success [6,8]. Wrangler [8], for example, provides a user-friendly, web-based interaction framework that helps non-professionals create data transformation programs composed of operations defined in the Potter's Wheel system [11].

SIGMOD'17, May 14-19, 2017, Chicago, IL, USA
 ISBN 978-1-4503-4197-4/17/05…$15.00
DOI: http://dx.doi.org/10.1145/3035918.3058732

Bureau of I.A.	
Regional Director	Numbers
Niles C.	Tel: (800)645-8397
	Fax: (907)586-7252
Jean H.	Tel: (918)781-4600
	Fax: (918)781-4604
Frank K.	Tel: (615)564-6500
	Fax: (615)564-6701

…

Table 1: A spreadsheet of business contact information

	Tel	Fax
Niles C.	(800)645-8397	(907)586-7252
Jean H.	(918)781-4600	(918)781-4604
Frank K.	(615)564-6500	(615)564-6701

…

Table 2: A relational form of Table 1

```
1. Split 'column 2' on ':'
2. Delete rows where 'column 3' is null
3. Fill split with values from above
4. Unfold 'column 2' on 'column 3'
```

Figure 1: The desired data transformation program

However, existing data transformation tools remain hard to use for two reasons: (1) ***High Skill***: Users are required to know the various transformation operations (which are often complicated and difficult to understand) and then decide what operations to use and in what order. (2) ***High Effort***: The amount of user effort increases as the length of the data transformation program grows.

Take the raw business contact information data in Table 1 as an example. The program shown in Figure 1 uses some operators from Potter's Wheel [11] to transform the raw input data to the relational form shown in Table 2. Composing this program with Wrangler will show both of the above usability issues. The steps the Wrangler user must take (illustrated in Figure 2) require the user to provide a healthy amount of information: (1) the correct operators, with (2) parameters for each operator (e.g., which rows the Delete operator should remove), (3) in the correct program order. Note that the user must provide this information even when the transformation program is relatively straightforward and "boring" (High Effort). In some cases, discerning the right parameterization can be genuinely intellectually challenging; for example, the

Unfold operation in Step 4 "unflattens" tables and moves information from data values to column names; this step is difficult for naïve users to grasp (High Skill).

To address these usability issues, we developed a Programming By Example (PBE) data transformation program synthesizer, FOOFAH. Our system is designed to be used by people without any programming background and to require minimal user effort. Unlike Wrangler, which asks the end user for procedural hints, our system allows the user to describe a desired transformation simply by specifying an input-output example: the user only needs to know how to *describe the transformed data*, as opposed to knowing any particular transformation operation that must be performed. Moreover, the amount of user effort that our system requires does not scale with the length of the desired transformation program.

Previous PBE-based data transformation program synthesizers [2, 5, 9] have been limited in their expressiveness. FlashRelate [2] and ProgFromEx [5] can only express *layout transformations* (i.e., changing data cells locations), while FlashExtract [9] only supports *syntactic transformation* (i.e., manipulating strings). Our working demonstration system will show that FOOFAH is effective in solving complex data transformation tasks, which are mixtures of both syntactic transformations and layout transformations. It requires fewer iterations and less interaction time than competing methods. Conference attendees will be able to interact with the working FOOFAH system, using either their own custom data or data that we provide.

We will now introduce the design of FOOFAH including the user input, algorithms, and architecture in Section 2. We will discuss the demonstration experience in Section 3.

2. SYSTEM FRAMEWORK

In this section, we explain the user input, the system output, our proposed solution, the user interaction model, and the system performance.

2.1 User Input

We require the user to describe the expected transformation through input-output examples. An input-output example $\mathcal{E}$ consists of a pair of datasets (e_i, e_o), where e_i is the *input example* sampled from the raw data to be transformed and e_o is the *output example* of what e_i should be transformed into after being processed by the data transformation program. The raw data is a grid of values (i.e., a spreadsheet). It can be non-relational, but should follow some regular structures. We assume the desired output is always a relational table. Further, we assume the input data can be transformed without any extra semantic information, such as knowing that *MI* is an abbreviation for *Michigan*. Other researchers have focused on composing dictionaries to address this latter problem [1, 3, 13]

The effectiveness of FOOFAH relies on the *fidelity* and *representativeness* of the user input. (1) The user needs to be *faithful* to the desired transformation while specifying $\mathcal{E}$, and not make any mistakes (e.g., typos or copy-paste mistakes). Otherwise, FOOFAH terminates the synthesis process early, allowing the user to fix any errors. (2) User-specified input-output example $\mathcal{E}$ should be *representative* enough to describe the desired transformation. In principle, the sample may need to be large, but we observed in our experiments that a small example made from two to three data records will almost always suffice.

Numbers
Tel: (800)645-8397
Fax: (907)586-7252
...

Table 3: The second column of Table 1

Numbers	
Tel	(800)645-8397
Fax	(907)586-7252
...	

Table 4: Split the second column of Table 1 by ':'

2.2 Synthesized Program

Our desired transformation programs convert raw data into a relational form. Such transformations usually require structural reorganizations or reshaping of the table cells and sometimes modification of the cell values themselves. We use some operators from the Potter's Wheel system [11], including Split, Drop, Move, Copy, Merge, Fold, Unfold, Fill, Divide, Delete, Extract and Transpose. This is a set of general-purpose data transformation operators expressive enough to accomplish many real data transformation tasks. A typical operator is Split, parameterized by a column number and a delimiter. When applying Split by ':' to the second column of Table 1 (shown in Table 3), the output is Table 4.

All of our data transformation operators take a whole table as input and output a new table. A good transformation program applies a sequence of appropriately-parameterized operators, repeatedly yielding a table that is closer to the desired output. The resulting transformation program is a loop-free or straight-line program, which is a program structure that has been successfully applied in many previous efforts in data transformation [5, 8, 14].

We formally define the data transformation to be synthesized as a loop-free series of operations $p_1, p_2, ..., p_k$ such that: (1) each operation $p_i = (op_i, par_1, \dots) : t_{\text{in}} \rightarrow t_{\text{out}}$. p_i includes operator op_i with corresponding parameter(s) and transforms input data table t_{in} to output data table t_{out}; (2) the output of operation p_i is the input of p_{i+1}.

2.3 Our Approach

We formulate our program synthesis problem as a search problem in a state space graph $G(V, A)$, where arcs represent potential data transformation operations, vertices represent intermediate states of the data as transformed by the operation on previously traversed arcs, e_i is the initial state of G, and e_o is the goal state.

To efficiently solve this search problem, we applied the well-known A* search algorithm. The A* algorithm requires a heuristic function that estimates the cost of the cheapest path from the current state to the goal, which it uses to determine the ultimate lowest-cost path through the search space. In our problem, the search path is made up of a series of Potter's Wheel operations. Thus, for the A* heuristic function, we use "Potter's Wheel distance" (i.e., the minimum number of Potter's Wheel operations required to transform the current intermediate state of the data to the goal state). One possible approach to measure "Potter's Wheel distance" is to create a rule-based heuristic to estimate the operators possibly needed. However, this is impractical in that (1) the rule-base heuristic is less reliable when multiple data transformation operations are used in one task, especially when using operations that drastically change the layout of the data, and (2) the rule-base heuristic may not apply when new data transformation operators are introduced. Thus, we developed a method that estimates "Potter's Wheel distance" without the necessity of guessing the actual operators used.

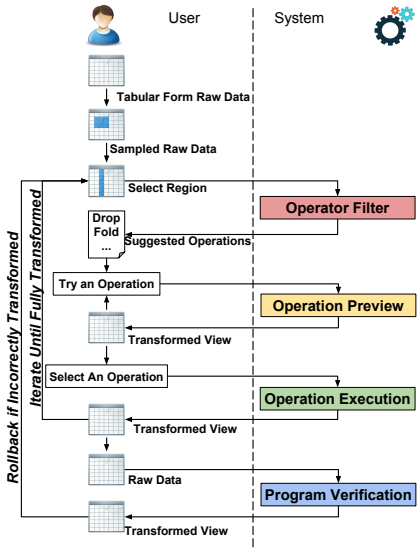

Figure 2: Wrangler architecture

We observed that each Potter's Wheel data transformation operator can be seen as a set of cell-level edit operations; the distinct cell-level operators are AddCell, RemoveCell, MoveCell, and TransformCell. For example, a Delete operation consists of a set of RemoveCell operations. Computing the cost of cell-level edit operations resembles the *graph edit distance* problem [12]. We hence created a novel concept, *Table Edit Distance* (TED), defined in (1), to measure the *distance* or *dissimilarity* between tables. TED is the minimum total cost of cell-level edit operations required to transform table T_1 to table T_2, where $\mathcal{P}(T_1, T_2)$ denotes the set of edit paths transforming T_1 to T_2 and $cost(e_i)$ is the cost of each cell-level edit operation e_i. We designed an efficient approximation algorithm to estimate TED (TED is equivalent to computing *graph edit distance*, which is NP-complete [4]).

$$TED(T_1, T_2) = \min_{(e_1,\ldots,e_k)\in\mathcal{P}(T_1,T_2)} \sum_{i=i}^{k} cost(e_i) \qquad (1)$$

Note that TED is not a suitable heuristic function for our problem since we actually wish to estimate Potter's Wheel distance, not the number of cell-level edit operations. We thus created a batch algorithm that reduces TED to an estimated Potter's Wheel distance. The batch algorithm groups the table edit operations in the edit path obtained from the TED calculation. This is inspired by our observation that Potter's Wheel operations tend to change horizontally or vertically adjacent cells. To further reduce the size of the graph and speed up the search, we created some operator-independent pruning rules using our domain knowledge. Thorough descriptions of our algorithms and methods are presented in our full paper [7].

2.4 System Architecture

Figure 3 depicts FOOFAH's architecture and user interaction workflow. FOOFAH saves the user from the complex interaction process in data transformation shown in Figure 2, only requiring a small amount of input from the user and simplifying the system architecture.

FOOFAH provides users with a web-based graphical interface with three major pages. The first page, **Example Creation**, is shown in the screenshot in Figure 4. It allows the user to provide an input-output example indicating the

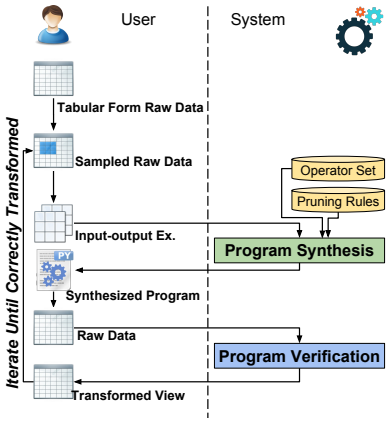

Figure 3: FOOFAH architecture

desired data transformation. The Input Example tabular form on the left is where the sampled raw data is specified. The user types in or copy-and-pastes a small sample of raw data, usually two to three records. In the Output Example tabular form on the right, the user enters the transformed view of the sample. The user presses the button on the bottom to trigger FOOFAH's synthesis process. The server-side Program Synthesis uses this input-output example in the A*-driven search process described in Section 2.3.

In our prototype, we set a 60-second time limit for each round of program synthesis. If FOOFAH successfully synthesizes a program, the program is displayed on the **Synthesized Program** page.[1] If FOOFAH encounters a timeout, it reports a failure. The synthesized program is guaranteed to transform the input example to the output example the user provides initially, but might fail to correctly transform the entire set of raw data, since the original sample may not be sufficiently representative. Without expertise in data transformation and programming, the user may not be able to verify the correctness of the program by just viewing it. To check whether the synthesized program is the desired one, the user may go to the **Verification** page, which has two tabular forms—Raw Data and Transformed Data—to input a new sample of the raw data, which could be as large as the entire raw data set. After the user hits the "Execute Program" button, the new data is sent along with the synthesized program to the Program Verification component on the server, which executes the program on the new sample. The system will return the transformed view of this new sample in the Transformed Data form, where the user can verify the correctness of the data transformation. If the program does not correctly transform the new data, the user can return to the **Example Creation** page to create a more representative example and start a new synthesis.

2.5 System Performance

Experiments (detailed in our full paper [7]) show that FOOFAH is able to efficiently synthesize high-quality data transformation programs with a small amount of user effort. In 86% of our benchmark tests, FOOFAH successfully synthesized programs in under five seconds using input-output examples composed of just three or fewer data records. Additionally, to test the impact of the size of user input, we

[1] FOOFAH currently returns a Python program to the user for use with its accompanying Python library, but the operations could be implemented in any suitable language.

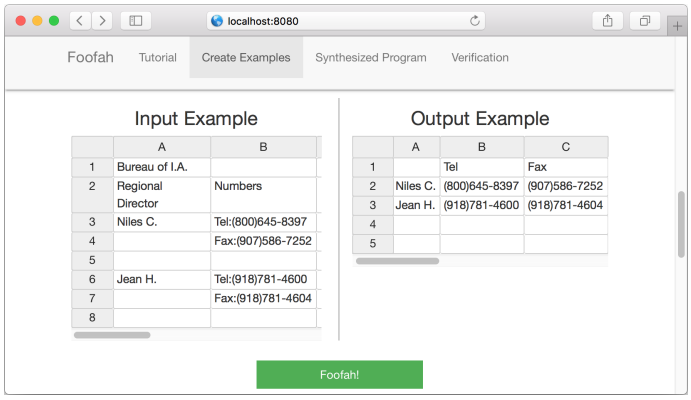

Figure 4: FOOFAH UI for specifying input-output examples

```
from Operations import *
# We hide the details of loading raw
    data for presentation purpose, and
    assume t is the loaded raw table
    represented by 2d list in python.

# split Column 1 on ':'
t = split(t, 1, ':')
# delete rows where Column 2 is null
t = delete(t, 2)
# fill Column 0 with value above
t = fill(t, 0)
# Unfold on Column 1
t = unfold(t, 1)
```

Figure 5: Synthesized program

varied the size of $\mathcal{E}$ from one to five records. We observed a near-linear increase of the synthesis time (average Pearson correlation of 0.75), though the synthesis time was still short in most cases (under five seconds in 72% of test cases with five records).

3. DEMONSTRATION

Our demonstration aims to show to the conference attendees that our proposed system FOOFAH is able to help a naïve user synthesize some real-world data transformation tasks and that it is generally easy to use.

To best illustrate how to use our tool, we collected three demonstration test cases: the motivating example in Section 1 and two other cases used by [8] and [5]. Attendees can also manually enter data.

To show how FOOFAH can help a user generate a correct data transformation program, we will use the motivating example from Section 1, where a user would do the following[2]:

1. Copy the first two records including the header information (lines 1-7) from the raw data in Table 1 and paste them into the Input Example form on the **Example Creation** page (Figure 4).
2. Specify the transformed view of the input example in the Output Example form on that same page:
   - 2.1. Type "Tel" and "Fax" as the column headers.
   - 2.2. Copy the human name "Niles C.", phone number "(800)645-8397", and fax number "(907)586-7252" from the Input Example form and paste them in second row of the Output Example form.
   - 2.3. Repeat the above process for the second record in the sampled raw data.
3. Hit the "Foofah!" button.

   FOOFAH *quickly infers the program shown in Figure 5, and displays it on the* ***Synthesized Program*** *page.*
4. To verify the program, copy the first three records (lines 1-10) of the raw data in Table 1 and paste them into the Test Data form in **Verification** page.
5. Hit the "Execute Program" button.

   The post-transformed view of the new sample shown to the user matches Table 2, enabling the user to verify the correctness of the synthesized program.

[2]A video can be found at https://youtu.be/Ura2pxez_Bo

4. CONCLUSION

Our demonstration showcases the ease-of-use of the proposed data transformation program synthesizer FOOFAH. With only a small example illustrating the desired transformation, FOOFAH can synthesize a high-quality program efficiently. This demonstration is a good illustration of our vision for a data transformation tool that requires minimal user effort and programming skill.

5. ACKNOWLEDGMENTS

This project is supported by National Science Foundation grants IIS-1250880, IIS-1054913, NSF IGERT grant 0903629, a Sloan Research Fellowship, and a CSE Dept. Fellowship.

6. REFERENCES

[1] Z. Abedjan, J. Morcos, I. Ilyas, M. Ouzzani, P. Papotti, and M. Stonebraker. DataXFormer: A robust transformation discovery system. In *ICDE*, 2016.
[2] D. W. Barowy, S. Gulwani, T. Hart, and B. Zorn. FlashRelate: Extracting relational data from semi-structured spreadsheets using examples. In *SIGPLAN*, 2015.
[3] Z. Chen, M. Cafarella, and H. Jagadish. Long-tail vocabulary dictionary extraction from the web. In *WSDM*. ACM, 2016.
[4] M. R. Gary and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-completeness*. WH Freeman and Company, New York, 1979.
[5] W. R. Harris and S. Gulwani. Spreadsheet table transformations from examples. In *ACM SIGPLAN Notices*, volume 46, pages 317–328. ACM, 2011.
[6] D. Huynh and S. Mazzocchi. Openrefine. http://openrefine.org, 2012.
[7] Z. Jin, M. R. Anderson, M. Cafarella, and H. V. Jagadish. Foofah: Transforming data by example. In *SIGMOD*, 2017.
[8] S. Kandel, A. Paepcke, J. Hellerstein, and J. Heer. Wrangler: Interactive visual specification of data transformation scripts. In *CHI*, 2011.
[9] V. Le and S. Gulwani. FlashExtract: A framework for data extraction by examples. In *ACM SIGPLAN Notices*, volume 49, pages 542–553. ACM, 2014.
[10] S. Lohr. For big-data scientists, janitor work is key hurdle to insights. *The New York Times*, 17, 2014.
[11] V. Raman and J. M. Hellerstein. Potter's Wheel: An interactive data cleaning system. In *VLDB*, 2001.
[12] A. Sanfeliu and K.-S. Fu. A distance measure between attributed relational graphs for pattern recognition. *IEEE Trans. on Systems, Man, and Cybernetics*, (3):353–362, 1983.
[13] R. Singh and S. Gulwani. Learning semantic string transformations from examples. *PVLDB*, 5(8):740–751, 2012.
[14] I. H. Witten and D. Mo. TELS: Learning text editing tasks from examples. In *Watch what I do*, pages 183–203. 1993.