Advanced Data Management (CSCI 490/680)

Data Transformation

Dr. David Koop
Wrangler

- Data cleaning takes a lot of **time** and **human effort**
- "Tedium is the message"
- Repeating this process on multiple data sets is even worse!
- Solution:
  - interactive interface (mixed-initiative)
  - transformation language with natural language "translations"
  - suggestions + "programming by demonstration"
Potter's Wheel: Example

[Table 1: Definitions of the various transforms.]

<table>
<thead>
<tr>
<th>Split</th>
<th>Merge</th>
<th>Drop</th>
<th>Add</th>
</tr>
</thead>
<tbody>
<tr>
<td>Anna</td>
<td>Davis</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Dole</td>
<td>Jerry</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Joan</td>
<td>Marsh</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Format

'(.*)', '(.*), (.*)' to '2 \1' [V. Raman and J. Hellerstein, 2001]
Potter's Wheel: Transforms

<table>
<thead>
<tr>
<th>Transform</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Format</td>
<td>( \phi(R, i, f) ) = { (a_1, \ldots, a_i-1, a_i+1, \ldots, a_n, f(a_i)) \mid (a_1, \ldots, a_n) \in R }</td>
</tr>
<tr>
<td>Add</td>
<td>( \alpha(R, x) ) = { (a_1, \ldots, a_n, x) \mid (a_1, \ldots, a_n) \in R }</td>
</tr>
<tr>
<td>Drop</td>
<td>( \pi(R, i) ) = { (a_1, \ldots, a_{i-1}, a_{i+1}, \ldots, a_n) \mid (a_1, \ldots, a_n) \in R }</td>
</tr>
<tr>
<td>Copy</td>
<td>( \kappa((a_1, \ldots, a_n), i) ) = { (a_1, \ldots, a_n, a_i) \mid (a_1, \ldots, a_n) \in R }</td>
</tr>
<tr>
<td>Merge</td>
<td>( \mu((a_1, \ldots, a_n), i, j, \text{glue}) ) = { (a_1, \ldots, a_{i-1}, a_i+1, \ldots, a_j-1, a_j+1, \ldots, a_n, a_i \oplus \text{glue} \oplus a_j) \mid (a_1, \ldots, a_n) \in R }</td>
</tr>
<tr>
<td>Split</td>
<td>( \omega((a_1, \ldots, a_n), i, \text{splitter}) ) = { (a_1, \ldots, a_{i-1}, a_i+1, \ldots, a_n, \text{left}(a_i, \text{splitter}), \text{right}(a_i, \text{splitter})) \mid (a_1, \ldots, a_n) \in R }</td>
</tr>
<tr>
<td>Divide</td>
<td>( \delta((a_1, \ldots, a_n), i, \text{pred}) ) = { (a_1, \ldots, a_{i-1}, a_i+1, \ldots, a_n, a_i, \text{null}) \mid (a_1, \ldots, a_n) \in R \wedge \text{pred}(a_i) } \cup \ { (a_1, \ldots, a_{i-1}, a_i+1, \ldots, a_n, \text{null}, a_i) \mid (a_1, \ldots, a_n) \in R \wedge \neg\text{pred}(a_i) }</td>
</tr>
<tr>
<td>Fold</td>
<td>( \lambda(R, i_1, i_2, \ldots i_k) ) = { (a_1, \ldots, a_{i_1-1}, a_{i_1}+1, \ldots, a_{i_2-1}, a_{i_2}+1, \ldots, a_{i_k-1}, a_{i_k}+1, \ldots, a_n, a_{i_l}) \mid (a_1, \ldots, a_n) \in R \wedge 1 \leq l \leq k }</td>
</tr>
<tr>
<td>Select</td>
<td>( \sigma(R, \text{pred}) ) = { (a_1, \ldots, a_n) \mid (a_1, \ldots, a_n) \in R \wedge \text{pred}((a_1, \ldots, a_n)) }</td>
</tr>
</tbody>
</table>

**Notation:** \( R \) is a relation with \( n \) columns. \( i, j \) are column indices and \( a_i \) represents the value of a column in a row. \( x \) and \( \text{glue} \) are values. \( f \) is a function mapping values to values. \( x \oplus y \) concatenates \( x \) and \( y \). \( \text{splitter} \) is a position in a string or a regular expression, \( \text{left}(x, \text{splitter}) \) is the left part of \( x \) after splitting by \( \text{splitter} \). \( \text{pred} \) is a function returning a boolean.

[V. Raman and J. Hellerstein, 2001]
Interface

• Automated Transformation Suggestions
• Editable Natural Language Explanations

• Visual Transformation Previews
• Transformation History

[Figures and tables]

[S. Kandel et al., 2011]
Improvements in Prediction

Update suggestions when given more information

[Heer et al., 2015]
Differences with Extract-Transform-Load (ETL)

- **ETL:**
  - Who: IT Professionals
  - Why: Create static data pipeline
  - What: Structured data
  - Where: Data centers

- **"Modern Data Preparation":**
  - Who: Analysts
  - Why: Solve problems by designing recipes to use data
  - What: Original, custom data blended with other data
  - Where: Cloud, desktop

[J. M. Hellerstein et al., 2018]
Handling Missing Data

- Filtering out missing data:
  - Can choose rows or columns
- Filling in missing data:
  - with a default value
  - with an interpolated value
- In pandas:

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dropna</td>
<td>Filter axis labels based on whether values for each label have missing data, with varying thresholds for how much missing data to tolerate.</td>
</tr>
<tr>
<td>fillna</td>
<td>Fill in missing data with some value or using an interpolation method such as 'ffill' or 'bfill'.</td>
</tr>
<tr>
<td>isnull</td>
<td>Return boolean values indicating which values are missing/NA.</td>
</tr>
<tr>
<td>notnull</td>
<td>Negation of isnull.</td>
</tr>
</tbody>
</table>
Filtering and Cleaning Data

• Find duplicates
  - duplicated: returns boolean Series indicating whether row is a duplicate—first instance is not marked as a duplicate

• Remove duplicates:
  - drop_duplicates: drops all rows where duplicated is True
  - keep: which value to keep (first or last)

• Can pass specific columns to check for duplicates, e.g. check only key column
Replacing Values

- `fillna` is a special case
- What if -999 in our dataset was identified as a missing value?

```
In [61]: data
Out[61]:
   0    1.0
  1  -999.0
  2     2.0
  3  -999.0
  4 -1000.0
  5     3.0
dtype: float64
```

```
In [62]: data.replace(-999, np.nan)
Out[62]:
   0     1.0
  1   NaN
  2     2.0
  3   NaN
  4 -1000.0
  5     3.0
dtype: float64
```

- Can pass list of values or dictionary to change different values
String Transformation

• One of the reasons for Python's popularity is string/text processing

• `split(<delimiter>)`: break a string into pieces:
  - `s = "12,13, 14"
    slist = s.split(',') # ["12", "13", " 14"]`

• `<delimiter> . join([<str>])`: join several strings by a delimiter
  - `"::" . join(slist) # "12:13: 14"`

• `strip()`: remove leading and trailing whitespace
  - `[p.strip() for p in slist] # ["12", "13", "14"]`
String Transformation

• `replace(<from>,<to>)`: change substrings to another substring
• `upper()/lower()`: casing
• `index(<str>)`: find where a substring first occurs (Error if not found)
• `find(<str>)`: same as `index` but −1 if not found
• `startswith()/endswith()`: boolean checks for string occurrence
Assignment 2

- Due Friday
- Same data as A1, different version of the dataset
- Dealing with the raw data now
- Same questions as A1, but use pandas
- Potential answers sent via email
- CS680 students + some questions about problems with the data
Test 1

- Wednesday, February 17, 3:30pm-4:45pm Online (Blackboard)
- Includes much of the python content we have covered plus data, data cleaning, data transformation topics (content through today's lecture)
- Format:
  - Multiple Choice
  - Free Response (see web page for examples)
  - CS680 students will have additional questions
- Coding questions will focus on broad syntax not your memorization of every pandas function
- Concept questions can include discussions of the research papers
Regular Expressions in Python

• import re
• re.search(<pattern>, <str_to_check>)
  - Returns None if no match, information about the match otherwise

• Capturing information about what is in a string → parentheses
• (\d+)/\d+/\d+ will capture information about the month
• match = re.search(''(\d+)/\d+/\d+','12/31/2016')
  if match:
    match.group() # 12
• re.findall(<pattern>, <str_to_check>)
  - Finds all matches in the string, search only finds the first match
• Can pass in flags to alter methods: e.g. re.IGNORECASE
Pandas String Methods

• Any column or series can have the string methods (e.g. replace, split) applied to the entire series
• Fast (vectorized) on whole columns or datasets
• use .str.<method_name>
• .str is important!

- data = pd.Series({'Dave': 'dave@google.com',
                  'Steve': 'steve@gmail.com',
                  'Rob': 'rob@gmail.com',
                  'Wes': np.nan})

  data.str.contains('gmail')
  data.str.split('@').str[1]
  data.str[-3:]
Regular Expression Methods

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>findall</td>
<td>Return all non-overlapping matching patterns in a string as a list</td>
</tr>
<tr>
<td>finditer</td>
<td>Like findall, but returns an iterator</td>
</tr>
<tr>
<td>match</td>
<td>Match pattern at start of string and optionally segment pattern components into groups; if the pattern matches, returns a match object, and otherwise None</td>
</tr>
<tr>
<td>search</td>
<td>Scan string for match to pattern; returning a match object if so; unlike match, the match can be anywhere in the string as opposed to only at the beginning</td>
</tr>
<tr>
<td>split</td>
<td>Break string into pieces at each occurrence of pattern</td>
</tr>
<tr>
<td>sub, subn</td>
<td>Replace all (sub) or first n occurrences (subn) of pattern in string with replacement expression; use symbols \1, \2, ... to refer to match group elements in the replacement string</td>
</tr>
</tbody>
</table>

Table 7-4. Regular expression methods

There is much more to regular expressions in Python, most of which is outside the book’s scope. Table 7-4 provides a brief summary.

Vectorized String Functions in pandas

Cleaning up a messy dataset for analysis often requires a lot of string munging and regularization. To complicate matters, a column containing strings will sometimes have missing data:
Pandas String Methods with Regexs

In [172]: pattern
Out[172]: '([A-Z0-9._%+-]+)@([A-Z0-9.-]+)\.(\[A-Z\][2,4])'

In [173]: data.str.findall(pattern, flags=re.IGNORECASE)
Out[173]:
Dave       [(dave, google, com)]
Rob        [(rob, gmail, com)]
Steve      [(steve, gmail, com)]
Wes        NaN
dtype: object

In [174]: matches = data.str.match(pattern, flags=re.IGNORECASE)

In [175]: matches
Out[175]:
Dave       True
Rob        True
Steve      True
Wes        NaN
dtype: object

[W. McKinney, Python for Data Analysis]
Foofah: Transforming Data By Example

Z. Jin, M. R. Anderson, M. Cafarella, and H. V. Jagadish
Foofah Discussion
Foofah Discussion

• What is the paper's contribution?
Foofah Discussion

• What is the paper's contribution?
• What questions do you have about what is going on?
Foofah Discussion

• What is the paper's contribution?
• What questions do you have about what is going on?
• What does the technique do well/have issues with?
Foofah Discussion

- What is the paper's contribution?
- What questions do you have about what is going on?
- What does the technique do well/have issues with?
- How does its approach compare with Trifacta?
Starting Point: Raw Data

![Table](image_url)

[Guo et al., 2011]
Goal

- Focus on data transformation
- Data transformation tools suffer usability issues:
  - High Skill: familiarity with operations and the effect or their order
  - High Effort: user effort increases as the program becomes longer
- Repetitive and tedious
- Goal: minimize a user's effort and reduce the required background knowledge for data transformation tasks
Getting Lost in Transformations

<table>
<thead>
<tr>
<th>Bureau of I.A.</th>
<th>Numbers</th>
</tr>
</thead>
<tbody>
<tr>
<td>Regional Director</td>
<td>Tel: (800)645-8397</td>
</tr>
<tr>
<td></td>
<td>Fax: (907)586-7252</td>
</tr>
<tr>
<td>Niles C.</td>
<td>Tel: (918)781-4600</td>
</tr>
<tr>
<td></td>
<td>Fax: (918)781-4604</td>
</tr>
<tr>
<td>Jean H.</td>
<td>Tel: (615)564-6500</td>
</tr>
<tr>
<td></td>
<td>Fax: (615)564-6701</td>
</tr>
<tr>
<td>Frank K.</td>
<td>Tel: (615)564-6500</td>
</tr>
<tr>
<td></td>
<td>Fax: (615)564-6701</td>
</tr>
</tbody>
</table>

Original Table

Problem Table

Split+Delete

Unfold

Intermediate Table

Fill+Unfold

Desired Solution

[Z. Jin et al., 2017]
Foofah Design: Programming by Example

Input-output Example

System

Synthesized Program

Raw Data

Input Example

Output Example

Test Data

Transformed Data

[Z. Jin et al., 2017]
Input, Output, and Transformations

Raw Data:
- A grid of values, i.e., spreadsheets
- “Somewhat” structured - must have some regular structure or is automatically generated.

User Input:
- Sample from raw data
- Transformed view of the sample

Program to synthesize:
- A loop-free Potter’s Wheel [2] program

Transformations Targeted:
1. Layout transformation
2. String transformation

[05-16-2017] 05/16/2017
[05-17-2017] 05/17/2017
...

[Z. Jin et al., 2017]
Transformations

<table>
<thead>
<tr>
<th>Operator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Drop</td>
<td>Deletes a column in the table</td>
</tr>
<tr>
<td>Move</td>
<td>Relocates a column from one position to another in the table</td>
</tr>
<tr>
<td>Copy</td>
<td>Duplicates a column and append the copied column to the end of the table</td>
</tr>
<tr>
<td>Merge</td>
<td>Concatenates two columns and append the merged column to the end of the table</td>
</tr>
<tr>
<td>Split</td>
<td>Separates a column into two or more halves at the occurrences of the delimiter</td>
</tr>
<tr>
<td>Fold</td>
<td>Collapses all columns after a specific column into one column in the output table</td>
</tr>
<tr>
<td>Unfold</td>
<td>“Unflatten” tables and move information from data values to column names</td>
</tr>
<tr>
<td>Fill</td>
<td>Fill empty cells with the value from above</td>
</tr>
<tr>
<td>Divide</td>
<td>Divide is used to divide one column into two columns based on some predicate</td>
</tr>
<tr>
<td>Delete</td>
<td>Delete rows or columns that match a given predicate</td>
</tr>
<tr>
<td>Extract</td>
<td>Extract first match of a given regular expression each cell of a designated column</td>
</tr>
<tr>
<td>Transpose</td>
<td>Transpose the rows and columns of the table</td>
</tr>
<tr>
<td>Wrap (added)</td>
<td>Concatenate multiple rows conditionally</td>
</tr>
</tbody>
</table>

Table 2: Data transformation operators used by state-of-art data transformation tool

<table>
<thead>
<tr>
<th>Operator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Drop</td>
<td>Deletes a column in the table</td>
</tr>
<tr>
<td>Move</td>
<td>Relocates a column from one position to another in the table</td>
</tr>
<tr>
<td>Copy</td>
<td>Duplicates a column and append the copied column to the end of the table</td>
</tr>
<tr>
<td>Merge</td>
<td>Concatenates two columns and append the merged column to the end of the table</td>
</tr>
<tr>
<td>Split</td>
<td>Separates a column into two or more halves at the occurrences of the delimiter</td>
</tr>
<tr>
<td>Fold</td>
<td>Collapses all columns after a specific column into one column in the output table</td>
</tr>
<tr>
<td>Unfold</td>
<td>“Unflatten” tables and move information from data values to column names</td>
</tr>
<tr>
<td>Fill</td>
<td>Fill empty cells with the value from above</td>
</tr>
<tr>
<td>Divide</td>
<td>Divide is used to divide one column into two columns based on some predicate</td>
</tr>
<tr>
<td>Delete</td>
<td>Delete rows or columns that match a given predicate</td>
</tr>
<tr>
<td>Extract</td>
<td>Extract first match of a given regular expression each cell of a designated column</td>
</tr>
<tr>
<td>Transpose</td>
<td>Transpose the rows and columns of the table</td>
</tr>
<tr>
<td>Wrap (added)</td>
<td>Concatenate multiple rows conditionally</td>
</tr>
</tbody>
</table>

[Z. Jin et al., 2017]
Proposed Solution

• Use a small, manually transformed portion of the data to infer a program (in Potter's Wheel syntax) based on the specified data transformation operations
• No loops
• Assumes relational tables
• … and perfect data?
Foofah Solution

A search problem solved by A* algorithm

edges: operation
nodes: different views of the data
A* search: iteratively explore the
node with min f(n)
f(n) = g(n) + h(n)

Input Example e_i
Delete       Unfold       Split       Unfold
Fold
Fill

Input Example e_o

estimated distance
observed distance

[D. Koop, CSCI 680/490, Spring 2021]
Our technique is motivated by the following study [1,3,4,5].

**Comparison:**
- ProgFromEx (pure)
- Foofah
- Wrangler
- POTTER'S WHEEL
- 50%
- 40%
- 30%
- 20%
- 10%
- 0%

**Success Rate:**
- ProgFromEx ≤ 5 sec
- Foofah ≤ 30 sec
- Wrangler
- POTTER'S WHEEL

**Failure Rate:**
- ProgFromEx 0%
- Foofah 0%
- Wrangler
- POTTER'S WHEEL

**Evolution:**
- ProgFromEx
- Foofah
- Wrangler
- POTTER'S WHEEL
- FlashRelate
- Progression of the 24th annual ACM symposium on User interface software and technology.

**Task Types:**
- String transformation
- Layout transformation

**Related Work:**
- Harris,
- Kandel
- ACM, 2011.
- Guo
- Potter's Wheel: An interactive data cleaning system.
Table Edit Distance

- Akin to Graph Edit Distance
- Count the number of operations required to transform one table to another
- Use Add/Remove/Modify + Move

Table Edit Distance (TED) Definition:
The cost of transforming Table $T_1$ to Table $T_2$ using the cell-level operators Add/Remove/Move/Transform cell.

$$TED(T_1, T_2) = \min_{(p_1, \ldots, p_k) \in P(T_1, T_2)} \sum_{i=0}^{k} \text{cost}(p_i)$$

- $P(T_1, T_2)$: Set of all “paths” transforming $T_1$ to $T_2$ using cell-level operators

[Z. Jin et al., 2017]
Table Edit Distance Batch

Batch the geometrically-adjacent cell-level operations of the same type

8 Transform operations

2 “batched” Transform operations

[2: Jin et al., 2017]
## Geometric Patterns Used to Batch

<table>
<thead>
<tr>
<th>Pattern</th>
<th>Formulation (X is a table edit operator)</th>
<th>Related Operators</th>
</tr>
</thead>
<tbody>
<tr>
<td>Horizontal to Horizontal</td>
<td>{X((x_i, y_i), (x_j, y_j)), X((x_i, y_i + 1), (x_j, y_j + 1)), \ldots}</td>
<td>Delete(Possibly)</td>
</tr>
<tr>
<td>Horizontal to Vertical</td>
<td>{X((x_i, y_i), (x_j, y_j)), X((x_i, y_i + 1), (x_j + 1, y_j)), \ldots}</td>
<td>Fold, Transpose</td>
</tr>
<tr>
<td>Vertical to Horizontal</td>
<td>{X((x_i, y_i), (x_j, y_j)), X((x_i + 1, y_i), (x_j, y_j + 1)), \ldots}</td>
<td>Unfold, Transpose</td>
</tr>
<tr>
<td>Vertical to Vertical</td>
<td>{X((x_i, y_i), (x_j, y_j)), X((x_i + 1, y_i), (x_j + 1, y_j)), \ldots}</td>
<td>Move, Copy, Merge, Split, Extract, Drop</td>
</tr>
<tr>
<td>One to Horizontal</td>
<td>{X((x_i, y_i), (x_j, y_j)), X((x_i + 1, y_i), (x_j + 1, y_j)), \ldots}</td>
<td>Fold(Possibly), Fill(Possibly)</td>
</tr>
<tr>
<td>One to Vertical</td>
<td>{X((x_i, y_i), (x_j, y_j)), X((x_i, y_i), (x_j + 1, y_j)), \ldots}</td>
<td>Fold, Fill</td>
</tr>
<tr>
<td>Remove Horizontal</td>
<td>{X((x_i, y_i)), X((x_i, y_i + 1)), \ldots}</td>
<td>Delete</td>
</tr>
<tr>
<td>Remove Vertical</td>
<td>{X((x_i, y_i)), X((x_i + 1, y_i)), \ldots}</td>
<td>Drop, Unfold</td>
</tr>
</tbody>
</table>

---

[Z. Jin et al., 2017]
Other Pruning Rules

- Global:
  - Missing Alphanumerics: check that character maintained
  - No effect: meaningless operation
  - Introducing Novel Symbols: check that no new characters added

- Property-specific:
  - Generating Empty Columns
  - Null in Column
Evaluation Results: # Test Records & Time

(a) Number of records required in test scenarios to infer *perfect* programs

(b) Worst and average synthesis time in each interaction

---

[Z. Jin et al., 2017]
5.4 Effectiveness of Pruning Rules

Figure 11: (a) and (b) show number of records and synthesis time required by different search strategies; (c) Percentage of tests synthesized in the experiments of Section 5.1; (c) Percentage of tests synthesized in the experiments of Section 5.1.

% of test cases

Time (seconds)

(a) Compare search strategies

(b) Effectiveness of pruning rules

NoPrune  PropPrune
GlobalPrune  FullPrune

% of test cases

Time (seconds)

[Z. Jin et al., 2017]
User Study Results

<table>
<thead>
<tr>
<th>Test</th>
<th>Complex</th>
<th>≥ 4 Ops</th>
<th>WRANGLER Time</th>
<th>WRANGLER Mouse</th>
<th>WRANGLER Key</th>
<th>FOOFAH Time</th>
<th>FOOFAH Mouse</th>
<th>FOOFAH Key</th>
</tr>
</thead>
<tbody>
<tr>
<td>PW1</td>
<td>No</td>
<td>No</td>
<td>104.2</td>
<td>17.8</td>
<td>11.6</td>
<td>49.4</td>
<td>20.8</td>
<td>22.6</td>
</tr>
<tr>
<td>PW3 (modified)</td>
<td>No</td>
<td>No</td>
<td>96.4</td>
<td>28.8</td>
<td>26.6</td>
<td>38.6</td>
<td>14.2</td>
<td>23.6</td>
</tr>
<tr>
<td>ProgFromEx13</td>
<td>Yes</td>
<td>No</td>
<td>263.6</td>
<td>59.0</td>
<td>16.2</td>
<td>145.8</td>
<td>43.6</td>
<td>78.4</td>
</tr>
<tr>
<td>PW5</td>
<td>Yes</td>
<td>No</td>
<td>242.0</td>
<td>52.0</td>
<td>15.2</td>
<td>58.8</td>
<td>31.4</td>
<td>32.4</td>
</tr>
<tr>
<td>ProgFromEx17</td>
<td>No</td>
<td>Yes</td>
<td>72.4</td>
<td>18.8</td>
<td>11.6</td>
<td>48.6</td>
<td>18.2</td>
<td>15.2</td>
</tr>
<tr>
<td>PW7</td>
<td>No</td>
<td>Yes</td>
<td>141.0</td>
<td>41.8</td>
<td>12.2</td>
<td>44.4</td>
<td>19.6</td>
<td>35.8</td>
</tr>
<tr>
<td>Proactive1</td>
<td>Yes</td>
<td>Yes</td>
<td>324.2</td>
<td>60.0</td>
<td>13.8</td>
<td>104.2</td>
<td>41.4</td>
<td>57.0</td>
</tr>
<tr>
<td>Wrangler3</td>
<td>Yes</td>
<td>Yes</td>
<td>590.6</td>
<td>133.2</td>
<td>29.6</td>
<td>137.0</td>
<td>58.6</td>
<td>99.8</td>
</tr>
</tbody>
</table>

[Z. Jin et al., 2017]
Comparisons with other tools

Success rates on pure layout transformation benchmark tasks

<table>
<thead>
<tr>
<th>Tool</th>
<th>Failure Rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>Foofah</td>
<td>12.60%</td>
</tr>
<tr>
<td>FlashRelate</td>
<td>2.30%</td>
</tr>
<tr>
<td>ProgFromEx</td>
<td>5.20%</td>
</tr>
<tr>
<td>Wrangler</td>
<td>0.00%</td>
</tr>
</tbody>
</table>

Success rates on benchmark tasks requiring syntactic transformations

<table>
<thead>
<tr>
<th>Tool</th>
<th>Failure Rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>Foofah</td>
<td>100.00%</td>
</tr>
<tr>
<td>FlashRelate</td>
<td>0.00%</td>
</tr>
<tr>
<td>ProgFromEx</td>
<td>4.30%</td>
</tr>
<tr>
<td>Wrangler</td>
<td>85.70%</td>
</tr>
</tbody>
</table>
TDE: Transform Data by Example

In a separate scenario, suppose one would like to identify the top sales day. This data in Figure 1 is clearly not ready for analysis yet – an analyst wanting to figure out which day-of-the-week has the most sales, for instance, cannot find it out on-the-fly. To help the analyst, we built TDE: Transform Data by Example.

TDE works like a search engine, which synthesizes programs consistent with the given examples, and return them as a ranked list within a few seconds. Hovering over the first program (using System.DateTime.Parse) gives a preview of all results (shaded in green).

For example, in the first column dates are represented in various forms. In the second column, first names (e.g., both the first two records will convert into "Doe, John"). This again requires complex transformations. The top program uses the System.DateTime.Parse method to convert the date column.

Transform Data by Example

Figure 2:

This figure shows an example of how TDE works. The user provides two desired output examples in column-D, along with a few input examples. Once she clicks on the "Get Suggestions" button, the front-end talks to a back-end service running on Microsoft Azure, which searches over thousands of indexed functions, synthetizes programs from a much larger search space (tens of millions), and returns a ranked list of programs consistent with the given examples, and return them as a ranked list within a few seconds. Hovering over the first program (using System.DateTime.Parse) gives a preview of all results (shaded in green).

Figure 3:

This figure shows additional examples for transformation for names. The first three values in column-D are provided as output examples. The desired first-names and last-names are marked in bold for ease of reading. A composed example is then provided as output. In the right part of Figure 2, a ranked list of programs is returned based on the user’s put/output examples. In Figure 2(left), a user provides two output examples to specify the desired output. Once she clicks on the "Get Suggestions" button, the front-end talks to a back-end service running on Microsoft Azure, which searches over thousands of indexed functions, synthetizes programs from a much larger search space (tens of millions), and returns a ranked list of programs consistent with the given examples, and return them as a ranked list within a few seconds. Hovering over the first program (using System.DateTime.Parse) gives a preview of all results (shaded in green).

Transform Data by Example

The top - ranked program uses the System.DateTime.Parse method to convert the date column.
TDE: Transform Data by Example

### Table: Address Transformation Examples

<table>
<thead>
<tr>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Address</strong></td>
<td><strong>Output</strong></td>
</tr>
<tr>
<td>4297 148th Avenue NE L105, Bellevue, WA 98007</td>
<td>Bellevue, WA, 98007</td>
</tr>
<tr>
<td>2720 N Mesa St, El Paso, 79902, USA</td>
<td>El Paso, TX, 79902</td>
</tr>
<tr>
<td>3524 W Shore Rd APT 1002, Warwick, 02886</td>
<td>Warwick, RI, 02886</td>
</tr>
<tr>
<td>4740 N 132nd St, Omaha, 68164</td>
<td>Omaha, NE, 68164</td>
</tr>
<tr>
<td>10508 Prairie Ln, Oklahoma City</td>
<td>Oklahoma City, OK, 73162</td>
</tr>
<tr>
<td>525 1st St, Marysville, WA 95901</td>
<td>Marysville, CA, 95901</td>
</tr>
<tr>
<td>211 W Ridge Dr, Waukon, 52172</td>
<td>Waukon, IA, 52172</td>
</tr>
<tr>
<td>1008 Whitlock Ave NW, Marietta, 30064</td>
<td>Marietta, GA, 30064</td>
</tr>
<tr>
<td>602 Highland Ave, Shinnston, 26431</td>
<td>Shinnston, WV, 26431</td>
</tr>
<tr>
<td>840 W Star St, Greenville, 27834</td>
<td>Greenville, NC, 27834</td>
</tr>
</tbody>
</table>

[Y. He et al., 2018]
TDE: Transform Data by Example

Figure 2: (Left): transformation for date-time. (Right): transformations for addresses.

In a separate scenario, suppose one would like identified problems like name parsing and address standardization to solve them in a variety of domains, and these domain-specific transformation problems often have solutions in the form of code libraries to solve them in a variety of domains, and they are not new – for decades developers have built custom programs to synthesize programs from a much larger search space (tens of thousands of functions). We develop novel algorithms to synthesize programs from a much larger search space (tens of thousands of functions). We develop novel algorithms to synthesize programs from a much larger search space (tens of thousands of functions). We develop novel algorithms to synthesize programs from a much larger search space (tens of thousands of functions). We develop novel algorithms to synthesize programs from a much larger search space (tens of thousands of functions).

The first two values are provided as output examples to produce city, state, and zip-code. Note that some of these info are missing from the input. A program invoking Bing Maps API is returned as the top result. The back-end service running on Microsoft Azure is an Excel add-in, currently in beta and available from Office Store [7]. From the Excel add-in, users can find transformations by providing a few input/output examples. In Figure 2(left), a user provides two examples predefined in a Domain Specific Language (DSL), as FlashFill that compose a small number of string primitives into a new program using the data in Figure 1. The desired first-names and last-names are marked in bold for ease of reading. A composed program using library CSharpNameParser from GitHub is returned. (Right): transformations for addresses.
TDE: Synthesized Function

<table>
<thead>
<tr>
<th>Input Examples</th>
<th>Return Object Dump</th>
<th>Member method result dump</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Year</td>
<td>Month</td>
</tr>
<tr>
<td>Wed, 12 Jan 2011</td>
<td>2011</td>
<td>01</td>
</tr>
<tr>
<td>Thu, 15 Sep 2011</td>
<td>2011</td>
<td>09</td>
</tr>
<tr>
<td>Mon, 17 Sep 2012</td>
<td>2012</td>
<td>09</td>
</tr>
<tr>
<td>2010-Nov-30 21:10:41</td>
<td>2010</td>
<td>11</td>
</tr>
<tr>
<td>2011-Jan-11 02:27:21</td>
<td>2011</td>
<td>01</td>
</tr>
<tr>
<td>2011-Jan-12</td>
<td>2011</td>
<td>01</td>
</tr>
</tbody>
</table>

For parameterization, in [32], we develop new algorithms based on a recursive program with right parameters as described above. We thus treat the two as a candidate pair.

Figures 13: Given that the target output is unique, because the intermediate tables shown in Figure 14, where the task is to transform input time in US western timezone, to US eastern timezone. This transformation would require not only using relevant fields and methods but also appropriate parameters (“+3 hours”). To test if the inverse relationship holds, we then invoke the appropriate member method corresponding to 3 hours. This leads to a new function returned by the L1 error in the day, month, and year, as shown in the figure. We then use to exhaustively invoke it as an “+3 hours” type and “memorize” all its possible values, respectively. We first identify functions discover concepts such as different purpose, which can be seen from the split (the substring after the following operations: We split each value using “,” and take the second component from the split (the substring after the first three characters of the input strings).

We illustrate it with an example below.

Example: Suppose the desired output is instead Target output for both input strings. Then with the first three characters of the target output for both input strings. We split each value using “,” and take the second component from the split (the substring after the following operations: We split each value using “,” and take the second component from the split (the substring after the following operations: We split each value using “,” and take the second component from the split (the substring after the first three characters of the input strings). This would produce the desired output for both input strings.

For parameterization, in [32], we develop new algorithms based on a recursive program with right parameters as described above. We thus treat the two as a candidate pair.

Figures 13: Given that the target output is unique, because the intermediate tables shown in Figure 14, where the task is to transform input time in US western timezone, to US eastern timezone. This transformation would require not only using relevant fields and methods but also appropriate parameters (“+3 hours”). To test if the inverse relationship holds, we then invoke the appropriate member method corresponding to 3 hours. This leads to a new function returned by the L1 error in the day, month, and year, as shown in the figure. We then use to exhaustively invoke it as an “+3 hours” type and “memorize” all its possible values, respectively. We first identify functions discover concepts such as different purpose, which can be seen from the split (the substring after the first three characters of the input strings). This would produce the desired output for both input strings.
TDE: Transform Data by Example

- Row-to-row translation only
- Search System, GitHub, and StackOverflow for functions
- Given dataset with examples
  - Use L1 from library
  - Compose synthesized programs (L2)
  - Rank best transformations
### TDE Benchmarks

<table>
<thead>
<tr>
<th>System</th>
<th>Total cases (239)</th>
<th>FF-GR-Trifacta (46)</th>
<th>Head cases (44)</th>
<th>StackOverflow (49)</th>
<th>BingQL-Unit (50)</th>
<th>BingQL-Other (50)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>TDE</strong></td>
<td>72% (173)</td>
<td>91% (42)</td>
<td>82% (36)</td>
<td>63% (31)</td>
<td>96% (48)</td>
<td>32% (16)</td>
</tr>
<tr>
<td><strong>TDE-NF</strong></td>
<td>53% (128)</td>
<td>87% (40)</td>
<td>41% (18)</td>
<td>35% (17)</td>
<td>96% (48)</td>
<td>10% (5)</td>
</tr>
<tr>
<td>FlashFill</td>
<td>23% (56)</td>
<td>57% (26)</td>
<td>34% (15)</td>
<td>31% (15)</td>
<td>0% (0)</td>
<td>0% (0)</td>
</tr>
<tr>
<td>Foofah</td>
<td>3% (7)</td>
<td>9% (4)</td>
<td>2% (1)</td>
<td>4% (2)</td>
<td>0% (0)</td>
<td>0% (0)</td>
</tr>
<tr>
<td>DataXFormer-UB</td>
<td>38% (90)</td>
<td>7% (3)</td>
<td>36% (16)</td>
<td>35% (17)</td>
<td>62% (31)</td>
<td>46% (23)</td>
</tr>
<tr>
<td>System-A</td>
<td>13% (30)</td>
<td>52% (24)</td>
<td>2% (1)</td>
<td>10% (5)</td>
<td>0% (0)</td>
<td>0% (0)</td>
</tr>
<tr>
<td>OpenRefine-Menü</td>
<td>4% (9)</td>
<td>13% (6)</td>
<td>2% (1)</td>
<td>4% (2)</td>
<td>0% (0)</td>
<td>0% (0)</td>
</tr>
</tbody>
</table>

- TDE and FlashFill focused on row-to-row transformations
- Foofah considers a wider range of transformations (table reformatting)

[Y. He et al., 2018]
TDE Benchmarks

<table>
<thead>
<tr>
<th>System</th>
<th>Total cases (239)</th>
<th>FF-GR-Trifacta (46)</th>
<th>Head cases (44)</th>
<th>StackOverflow (49)</th>
<th>BingQL-Unit (50)</th>
<th>BingQL-Other (50)</th>
</tr>
</thead>
<tbody>
<tr>
<td>TDE</td>
<td>72% (173)</td>
<td>91% (42)</td>
<td>82% (36)</td>
<td>63% (31)</td>
<td>96% (48)</td>
<td>32% (16)</td>
</tr>
<tr>
<td>TDE-NF</td>
<td>53% (128)</td>
<td>87% (40)</td>
<td>41% (18)</td>
<td>35% (17)</td>
<td>96% (48)</td>
<td>10% (5)</td>
</tr>
<tr>
<td>FlashFill</td>
<td>23% (56)</td>
<td>57% (26)</td>
<td>34% (15)</td>
<td>31% (15)</td>
<td>0% (0)</td>
<td>0% (0)</td>
</tr>
<tr>
<td>Foofah</td>
<td>3% (7)</td>
<td>9% (4)</td>
<td>2% (1)</td>
<td>4% (2)</td>
<td>0% (0)</td>
<td>0% (0)</td>
</tr>
<tr>
<td>DataXFormer-UB</td>
<td>38% (90)</td>
<td>7% (3)</td>
<td>36% (16)</td>
<td>35% (17)</td>
<td>62% (31)</td>
<td>46% (23)</td>
</tr>
<tr>
<td>System-A</td>
<td>13% (30)</td>
<td>52% (24)</td>
<td>2% (1)</td>
<td>10% (5)</td>
<td>0% (0)</td>
<td>0% (0)</td>
</tr>
<tr>
<td>OpenRefine-Menu</td>
<td>4% (9)</td>
<td>13% (6)</td>
<td>2% (1)</td>
<td>4% (2)</td>
<td>0% (0)</td>
<td>0% (0)</td>
</tr>
</tbody>
</table>

- TDE and FlashFill focused on row-to-row transformations
- Foofah considers a wider range of transformations (table reformatting)
Trifacta's Transform by Example