CIS 602-01: Scalable Data Analysis

Data Cleaning & Transformation

Dr. David Koop
Spark Project Goals

Extend the MapReduce model to better support two common classes of analytics apps:

- **Iterative** algorithms (machine learning, graphs)
- **Interactive** data mining

Enhance programmability:

- Integrate into Scala programming language
- Allow interactive use from Scala interpreter
Solution: Resilient Distributed Datasets (RDDs)

• Allow apps to keep working sets in memory for efficient reuse
• Retain the attractive properties of MapReduce
  - Fault tolerance, data locality, scalability
• Support a wide range of applications
# Spark Operations

<table>
<thead>
<tr>
<th>Transformations (define a new RDD)</th>
<th>map</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>filter</td>
</tr>
<tr>
<td></td>
<td>sample</td>
</tr>
<tr>
<td></td>
<td>groupByKey</td>
</tr>
<tr>
<td></td>
<td>reduceByKey</td>
</tr>
<tr>
<td></td>
<td>sortByKey</td>
</tr>
<tr>
<td>Actions (return a result to driver program)</td>
<td>flatMap</td>
</tr>
<tr>
<td></td>
<td>union</td>
</tr>
<tr>
<td></td>
<td>join</td>
</tr>
<tr>
<td></td>
<td>cogroup</td>
</tr>
<tr>
<td></td>
<td>cross</td>
</tr>
<tr>
<td></td>
<td>mapValues</td>
</tr>
<tr>
<td></td>
<td>collect</td>
</tr>
<tr>
<td></td>
<td>reduce</td>
</tr>
<tr>
<td></td>
<td>count</td>
</tr>
<tr>
<td></td>
<td>save</td>
</tr>
<tr>
<td></td>
<td>lookupKey</td>
</tr>
</tbody>
</table>

[Spark Overview]
Spark Job Stages

Example of how Spark computes job stages. Boxes with solid outlines are RDDs. Partitions are shaded rectangles, in black if they are already in memory. To run an action on RDD G, we build build stages at wide dependencies and pipeline narrow transformations inside each stage. In this case, stage 1’s output RDD is already in RAM, so we run stage 2 and then 3.

[Zaharia et al., 2012]
Spark 2.0

• Unified engine
  - Express entire workflow in one API
  - Connect existing libraries & storage
• High-level APIs with space to optimize
  - RDDs, DataFrames, ML pipelines

[M. Zaharia, 2016]
Spark Structured API Example

```
# DataFrame API
events = sc.read.json("/logs")
stats = events.join(users) .groupBy("loc","status") .avg("duration")
errors = stats.where(stats.status == "ERR")

# Optimized Plan
while(logs.hasNext) {
  e = logs.next
  if(e.status == "ERR") {
    u = users.get(e.uid)
    key = (u.loc, e.status)
    sum(key) += e.duration
    count(key) += 1
  }
}
...

# Specialized Code
[M. Zaharia, 2016]
```
Structured Streaming: Full Continuous Apps

Pure Streaming System

Input Stream → Streaming Computation → Output Sink

Continuous Application

Input Stream → Continuous Application → Output Sink

Static Data

consistent with

Batch Job

Ad-hoc Queries

End Goal: Full Continuous Apps

[M. Zaharia, 2016]
Spark Structured Streaming API

Example batch job:

```python
logs = ctx.read.format("json").open("hdfs://logs")
logs.groupBy("userid", "hour").avg("latency")
    .write.format("parquet")
    .save("s3://...")
```

Example as streaming:

```python
logs = ctx.readStream.format("json").load("hdfs://logs")
logs.groupBy("userid", "hour").avg("latency")
    .writeStream.format("parquet")
    .start("s3://...")
```

[M. Zaharia, 2016]
Final Project

• Option 1: Scalable analysis of an existing data set
  - Search for interesting datasets
  - Think about questions that would be interesting to answer
  - What scalability issues will you have to address?

• Option 2: Research project focusing on scalable analysis
  - If existing project, is it currently scalable? If not, what to change?
  - If new project, what scalability issues are you addressing?
  - Scalable toolkits are ok (e.g. notebook interface for mapboxgl)
Reading Response

• "SampleClean: Fast and Reliable Analytics on Dirty Data", S. Krishnan et al., 2015

• Due Thursday, Oct. 26 at 5pm

• Focus on critique but also make sure to include key contributions in the summary
Data Wrangling

- Remove errors, find inconsistencies (data cleaning)
- Reshape data (data transformation)
- Format data
- Deal with missing data
Wrangler: Interactive Visual Specification of Data Transformation Scripts

S. Kandel, A. Paepcke, J. Hellerstein, J. Heer
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"
Previous Work: Potter's Wheel

- V. Raman and J. Hellerstein, 2001
- Defines structure extractions for identifying fields
- Defines transformations on the data
- Allows user interaction
### Potter's Wheel: Structure Extraction

<table>
<thead>
<tr>
<th>Example Column Value (Example erroneous values)</th>
<th># Structures Enumerated</th>
<th>Final Structure Chosen (Punc = Punctuation)</th>
</tr>
</thead>
<tbody>
<tr>
<td>-60</td>
<td>5</td>
<td>Integer</td>
</tr>
<tr>
<td>UNITED, DELTA, AMERICAN etc.</td>
<td>5</td>
<td>IspellWord</td>
</tr>
<tr>
<td>SFO, LAX etc. (JFK to OAK)</td>
<td>12</td>
<td>AllCapsWord</td>
</tr>
<tr>
<td>1998/01/12</td>
<td>9</td>
<td>Int Punc(/) Int Punc(/) Int</td>
</tr>
<tr>
<td>M, Tu, Thu etc.</td>
<td>5</td>
<td>Capitalized Word</td>
</tr>
<tr>
<td>06:22</td>
<td>5</td>
<td>Int(len 2) Punc(:) Int(len 2)</td>
</tr>
<tr>
<td>12.8.15.147 (ferret03.webtop.com)</td>
<td>9</td>
<td>Double Punc(’) Double</td>
</tr>
<tr>
<td>”GET\b (\b)</td>
<td>5</td>
<td>Punc(””) IspellWord Punc()</td>
</tr>
<tr>
<td>/postmodern/lecs/xia/sld013.htm</td>
<td>4</td>
<td>ξ*</td>
</tr>
<tr>
<td>HTTP</td>
<td>3</td>
<td>AllCapsWord(HTTP)</td>
</tr>
<tr>
<td>/1.0</td>
<td>6</td>
<td>Punc(/) Double(1.0)</td>
</tr>
</tbody>
</table>

[Source: V. Raman and J. Hellerstein, 2001]
Potter's Wheel:

### Table: Transform Definitions

<table>
<thead>
<tr>
<th>Transform</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Format $\phi(R, i, f)$</td>
<td>${(a_1, \ldots, a_{i-1}, a_{i+1}, \ldots, a_n, f(a_i)) \mid (a_1, \ldots, a_n) \in R}$</td>
</tr>
<tr>
<td>Add $\alpha(R, x)$</td>
<td>${(a_1, \ldots, a_n, x) \mid (a_1, \ldots, a_n) \in R}$</td>
</tr>
<tr>
<td>Drop $\pi(R, i)$</td>
<td>${(a_1, \ldots, a_{i-1}, a_{i+1}, \ldots, a_n) \mid (a_1, \ldots, a_n) \in R}$</td>
</tr>
<tr>
<td>Copy $\kappa((a_1, \ldots, a_n), i)$</td>
<td>${(a_1, \ldots, a_n, a_i) \mid (a_1, \ldots, a_n) \in R}$</td>
</tr>
<tr>
<td>Merge $\mu((a_1, \ldots, a_n), i, j, glue)$</td>
<td>${(a_1, \ldots, a_{i-1}, a_i, a_{i+1}, \ldots, a_{j-1}, a_{j+1}, \ldots, a_n, a_i \odot glue \odot a_j) \mid (a_1, \ldots, a_n) \in R}$</td>
</tr>
<tr>
<td>Split $\omega((a_1, \ldots, a_n), i, splitter)$</td>
<td>${(a_1, \ldots, a_{i-1}, a_i, a_{i+1}, \ldots, a_n, \text{left}(a_i, \text{splitter}), \text{right}(a_i, \text{splitter})) \mid (a_1, \ldots, a_n) \in R}$</td>
</tr>
<tr>
<td>Divide $\delta((a_1, \ldots, a_n), i, \text{pred})$</td>
<td>${(a_1, \ldots, a_{i-1}, a_i, a_{i+1}, \ldots, a_n, a_i, \text{null}) \mid (a_1, \ldots, a_n) \in R \land \text{pred}(a_i)} \cup {(a_1, \ldots, a_n, \text{null}, a_i) \mid (a_1, \ldots, a_n) \in R \land \neg\text{pred}(a_i)}$</td>
</tr>
<tr>
<td>Fold $\lambda(R, i_1, i_2, \ldots, i_k)$</td>
<td>${(a_1, \ldots, a_{i_1-1}, a_{i_1+1}, \ldots, a_{i_2-1}, a_{i_2+1}, \ldots, a_{i_k-1}, a_{i_k+1}, \ldots, a_n, a_{i_1}) \mid (a_1, \ldots, a_n) \in R \land 1 \leq l \leq k}$</td>
</tr>
<tr>
<td>Select $\sigma(R, \text{pred})$</td>
<td>${(a_1, \ldots, a_n) \mid (a_1, \ldots, a_n) \in R \land \text{pred}((a_1, \ldots, a_n))}$</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 glue are values. $f$ is a function mapping values to values. $x \odot y$ concatenates $x$ and $y$. splitter is a position in a string or a regular expression, left($x$, splitter) is the left part of $x$ after splitting by splitter. pred is a function returning a boolean.

[V. Raman and J. Hellerstein, 2001]
### Potter's Wheel: Example

**Format**

- `'(.*)', (.*)' to \'2 \1'`

#### Example Table

<table>
<thead>
<tr>
<th></th>
<th>Stewart, Bob</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Anna</td>
<td>Davis</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Dole, Jerry</td>
<td></td>
</tr>
<tr>
<td>Joan</td>
<td>Marsh</td>
<td></td>
</tr>
</tbody>
</table>

- **Bob Stewart**
  - Anna Davis
  - Jerry Dole
  - Joan Marsh

- **Split at ' '**
  - 2 Merges
    - Anna Davis
    - Jerry Dole
    - Joan Marsh

- **2 Merges**
  - Bob Stewart
    - Anna Davis
    - Jerry Dole
    - Joan Marsh

---

[V. Raman and J. Hellerstein, 2001]
Potter's Wheel: Inferring Structure from Examples

| Example Values Split By User (| is user specified split position) | Inferred Structure | Comments |
|--------------------------------|------------------------|-------------------|
| Taylor, Jane | $52,072 Blair, John | $73,238 Tony Smith | $1,00,533 | (<ξ*> < ',' Money >) | Parsing is doable despite no good delimiter. A regular expression domain can infer a structure of $[0-9,]*$ for last component. |
| MAA | to | SIN JFK | to | SFO LAX | -- | ORD SEA | / | OAK | (<len 3 identifier> < ξ* > < len 3 identifier >) | Parsing is possible despite multiple delimiters. |
| 321 Blake #7 | , | Berkeley | , | CA 94720 719 MLK Road | , | Fremont | , | CA 95743 | (<number ξ*> < ',' word > < ',' (2 letter word) (5 letter integer)>) | Parsing is easy because of consistent delimiter. |

[V. Raman and J. Hellerstein, 2001]
Wrangler Transformation Language

- Based on Potter's Wheel
- Map: Delete, Extract, Cut, Split, Update
- Lookup/join: Use external data (e.g. from zipcode→state)
- Reshape: Fold and Unfold (aka pivot)
- Positional: Fill and lag
- Sorting, aggregation, key generation, schema transforms
Interface

- Automated Transformation Suggestions
- Editable Natural Language Explanations
- Fill Bangladesh by copying values from above
- Fill Bangladesh by averaging the values from above
- Visual Transformation Previews
- Transformation History

[Image of interface with examples of filling and averaging]

[S. Kandel et al., 2011]
Automation from past actions

- Infer parameter sets from user interaction
- Generating transforms

(a) Reported crime in Alabama

(b) selection: {'Alabama'} 'in' → {'in', word, lowercase}

(c) selection: {'Alabama'}, (word)

(d) selection: {'Alabama'}, (word)

(e) selection: {'Alabama'}, (word)

- Ranking and ordering transformations:
  - Based on user preferences, difficulty, and corpus frequency
  - Sort transforms by type and diversify suggestions
Evaluation

• Compare with Excel
• Tests:
  - Extract text from a single string entry
  - Fill in missing values with estimates
  - Reshape tables
• Allowed users to ask questions about Excel, not Wrangler
• Found significant effect of tool and users found previews and suggestions helpful
• Complaint: No manual fallback, make implications of user choices more obvious for users
COMPARATIVE EVALUATION WITH EXCEL

We divided subjects into “novices” and “experts” according to their prior experience with Excel on a 10-point scale (1 being novices). The median score for Excel novices was 3.0 and that for Excel experts was 6.5. We acknowledge that this is not an ideal cleaning solution for the data set, but it nonetheless served as a useful test.

We recruited 12 participants, all professional analysts or graduates of professional programs. Subjects rated their prior experience with Excel on a 10-point scale (1 being novices). Their prior experience was 5.5 on average.

Participants and Methods

We randomized the presentation of tasks and tools across subjects. In each task, we asked subjects to transform a data set into a new format, presented them as a picture of the final data table, and evaluated students who regularly work with data. Subjects rated their prior experience with Excel on a 10-point scale (1 being novices).

Data cleaning tasks

Our post-study questionnaire asked users to rate automated suggestions, visual previews, and direct editing of transforms. We divided subjects into “novices” and “experts” according to their prior experience with Excel on a 10-point scale (1 being novices). The median score for Excel novices was 3.0 and that for Excel experts was 6.5.

Three common data cleaning tasks: value extraction, missing value imputation, and table reshaping. Our goal was to compare task completion times and observe data cleaning strategies for different tasks. These strategies were largely consistent across users. For text selection, users frequently used word-based selection tools. For numeric data, users used a combination of number-based selection and direct editing.

For value extraction, users often selected values that were close to the desired range but not exactly equal. For example, users would select values that were 100% larger than the desired value. For table reshaping, users used a combination of word-based selection and direct editing.

Task Completion Times

Figure 11. Task completion times. Black bars indicate median values. Wrangler Accelerates Transform Specification

Median Wrangler performance is over twice as fast in all tasks. The effect of tool was highly significant (F(1, 54) = 23.65, p < 0.001). No other interactions were significant. Users rated previews for comparison: all subjects use it regularly and half self-report as experts. Excel also supports our chosen tasks. No other tools were used. Users completed the clean-up tasks significantly more quickly with Wrangler than with Excel (Fig. 11).

Figure 11. Task completion times. Black bars indicate median values.

[Figure 11: Task completion times]

We presented a 10 minute Wrangler tutorial describing basic knowledge and 10 being expert); the median score for Excel novices was 3.0 and that for Excel experts was 6.5.

We found a significant interaction effect of task (T1, T2, T3).


times with task, tool, and Excel novice/expert

Task Completion Time (minutes)

0 1 2 3 4 5 6 7 8 9 10

Wrangler Excel

User Study Task Completion Time (minutes)

Figure 11. Task completion times. Black bars indicate median values.

T1 T2 T3

[Figure 11: Task completion times]

We divided subjects into “novices” and “experts” according to their prior experience with Excel on a 10-point scale (1 being novices). The median score for Excel novices was 3.0 and that for Excel experts was 6.5.

[Figure 11: Task completion times]

We divided subjects into “novices” and “experts” according to their prior experience with Excel on a 10-point scale (1 being novices). The median score for Excel novices was 3.0 and that for Excel experts was 6.5.
Data Wrangler Demo

- [http://vis.stanford.edu/wrangler/app/](http://vis.stanford.edu/wrangler/app/)

```
<table>
<thead>
<tr>
<th>Transform Script</th>
<th>Import</th>
<th>Export</th>
</tr>
</thead>
<tbody>
<tr>
<td>Split data repeatedly on newline into rows</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Split split repeatedly on ','</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Promote row 0 to header</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Year</th>
<th>Property_crime_rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Reported crime in Alabama</td>
</tr>
<tr>
<td>1</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>2004</td>
</tr>
<tr>
<td>3</td>
<td>2005</td>
</tr>
<tr>
<td>4</td>
<td>2006</td>
</tr>
<tr>
<td>5</td>
<td>2007</td>
</tr>
<tr>
<td>6</td>
<td>2008</td>
</tr>
<tr>
<td>7</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>Reported crime in Alaska</td>
</tr>
<tr>
<td>9</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>2004</td>
</tr>
<tr>
<td>11</td>
<td>2005</td>
</tr>
<tr>
<td>12</td>
<td>2006</td>
</tr>
</tbody>
</table>
```

Delete row 7

Delete empty rows

Fill row 7 by copying values from above
Foofah: Transforming Data By Example

Z. Jin, M. R. Anderson, M. Cafarella, and H. V. Jagadish
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
Figure 1: A spreadsheet of business contact information

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

[22] Figure 2. Manually transforming the data record-by-record.

The many fields that depend on data for decision making present a challenge: before any analysis can be done, data from a variety of sources must be transformed into a format that is appropriate for the analysis tools. This data transformation task is tedious, requires about 20-30 minutes of work, and is not always performed well. Current commercial software packages often charge exorbitant fees to perform data transformation, and require users to be experts in programming. These tools are not particularly user-friendly: they don’t transform the intermediate table correctly, since raw data are transformed exactly as desired. Non-savvy users may find such conditional usage of software packages too expensive, especially for data transformation (Section 4).

Consider another scenario where the same task becomes more complex. Bob intends to perform a cross-tabulation operation that requires about 40-60 minutes of work. He discovers that the software package he is using does not transform the intermediate table correctly, since raw data are transformed exactly as desired. Bob finds the conditional usage of software packages too expensive, especially for data transformation. He is frustrated and considers giving up.

To resolve the above usability issues, we envision a data transformation tool that is fully automatic. This tool allows the user to specify a desired transformation simply by providing a stream of input-output examples. The synthesized program is designed to be easy-to-understand (it is a straight-line program comprised of simple primitives), so an unsophisticated user can understand the program without being concerned with the transformation steps required to get there. We implemented our technique in a program synthesizer called Wrangler.

The problem of transforming unstructured data—a task that is often labor-intensive and tedious. The requirement for programming data transformation operations to generate a program that, given the input to the function, produces the correct output is designed to address. In general, we aim to transform a pair, without being concerned with the transformation steps.

We explored Related Work in Section 6 and finished with a conclusion in Section 7.
Foofah Design: Programming by Example

Input-output Example

Raw Data

System

Synthesized Program

Test Data

Transformed Data

Raw Data

Input Example

Output Example

Synthesized Data Transformation Program in Python

[D. Koop, CIS 602-01, Fall 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

[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>

[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?
Our goal: minimize the amount of user effort. Wrangler\[1\] often require repetitive and tedious work and a transformation using Excel requires too much user effort.

1. Input

- Program to synthesize:
- Raw Data:

A regular structure or is automatically generated. "Somewhat" grid of values, i.e., spreadsheets often requires a user's effort and reduce the required view.

Structured Raw Data System

Most data transformation operations can be seen as many level transformation operations. The Table $T^1$, $T^2$, ..., $T^n$ can be seen as the $T^0$ to $T^n$ paths. Some of the operations applied to each path are:

- Add/Remove/Move/Transform
- Distance

Input Examples

A search problem solved by A* algorithm

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

$$f(n) = g(n) + h(n)$$

estimated distance
observed distance

example

[Z. Jin et al., 2017]
Most transformations are composed of cell-based operations

- **Add a cell**
- **Remove a cell**
- **Move a cell**
- **Transform a cell**
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.

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

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

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

- **8 Transform operations**
  - Tel: (800)645-8397
  - Fax: (907)586-7252
  - Tel: (918)781-4600
  - Fax: (918)781-4604

- **2 “batched” Transform operations**
  - Tel: (800)645-8397
  - Fax: (907)586-7252
  - Tel: (918)781-4600
  - Fax: (918)781-4604
# 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, y_i), (x_j, y_j + 1)), \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>
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
pruning" bad states, by giving them low priority in search. In fact, if we look at "BFS NoPrune" and "BFS" in Figure 12a, the di-rective these pruning rules boost the search efficiency of the program synthesis. However, the difference between their response time is quite significant, indicating that our proposed strategy, TED Batch, is effective.

5.4 Effectiveness of Pruning Rules

We examined the effectiveness of pruning rules for data transformation. We calculate the time required to synthesize the programs with and without these pruning rules to achieve the highest success rate of any of the strategies, as shown in Figure 12c.

Figure 12 presents the response times of the search strategies and pruning rules. Figure 12a shows the time required to synthesize the programs with and without pruning rules. The TED Batch fared compared to other search strategies on complex data transformations, we wished to know how TED outperforms the other three strategies.

In Section 5.1, we chose eight user study tasks of varied length and complexity. We asked participants to work on both test cases. This is evidence that the system can be improved through the addition of new operators, which can be easily incorporated without rewriting the core algorithm. This is shown in Figure 12b, which presents the worst and average synthesis time in each interaction.

5.6 User Effort Study

A property of our program synthesis technique is its operator-independence, as we discussed in Section 4. To demonstrate this, we compared the effectiveness of pruning rules with and without a newly added operator:Wrap every column \( W_1 \), Wrap on row \( W_2 \), and Wrap on all \( W_1 \& W_2 \). Column "Wrap yes" indicates if a task requires a data transformation program with 4 or more operations. Figure 12c shows the response time of all new variants of Wrap, using the same set of test cases as in Section 5.3. The addition of the new variants of Wrap, using the same set of test cases as in Section 5.3, increases the synthesis time of overall test cases did not increase. This indicates that the effort required by both systems.

Before the experiment, we educated participants on how to ask participants to work on both test cases. This is evidence that the system can be improved through the addition of new operators, which can be easily incorporated without rewriting the core algorithm. This is shown in Figure 12b, which presents the worst and average synthesis time in each interaction.

Figure 12c shows the response time of the search strategies and pruning rules. Since end users often feel frustrated when handling complex data transformations, we wished to know how TED outperforms the other three strategies.
### 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 vs Wrangler</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 52.6%</td>
<td>20.8</td>
<td>22.6</td>
<td></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 60.0%</td>
<td>14.2</td>
<td>23.6</td>
<td></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 44.7%</td>
<td>43.6</td>
<td>78.4</td>
<td></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 75.7%</td>
<td>31.4</td>
<td>32.4</td>
<td></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 32.9%</td>
<td>18.2</td>
<td>15.2</td>
<td></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 68.5%</td>
<td>19.6</td>
<td>35.8</td>
<td></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 67.9%</td>
<td>41.4</td>
<td>57.0</td>
<td></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 76.8%</td>
<td>58.6</td>
<td>99.8</td>
<td></td>
</tr>
</tbody>
</table>
Comparisons with other tools

Success rates on pure layout transformation benchmark tasks

Success rates on benchmark tasks requiring syntactic transformations
Discussion
Pandas Transformations

- Split: str.split
- Fold/Unfold: stack/unstack
- Merge, join, and concatenate documentation:
Reading Response

• "SampleClean: Fast and Reliable Analytics on Dirty Data", S. Krishnan et al., 2015
• Due Thursday, Oct. 26 at 5pm
• Focus on critique but also make sure to include key contributions in the summary