Explaining the outputs of Modern Data Analytics

Zaheer Chothia, John Liagouris, Frank McSherry, Timothy Roscoe

work done at the Systems Group, ETH Zürich
Modern Analytics

Modern Analytics Computation
Modern Analytics

Modern Analytics Computation
Modern Analytics

Modern Analytics
Computation
Modern Analytics

We can do pretty amazing computations.
Modern Analytics
Explaining the outputs of Modern Analytics Computation
Explaining the outputs of Modern Analytics
Explaining the outputs of Modern Analytics
Explaining the outputs of Modern Analytics
Explaining the outputs of Modern Analytics

For specified outputs, identify a subset of the inputs which produce these outputs.

(producing additional output is totally fine.)
Explaining the outputs of Modern Analytics
Prior research on the subject of
Explaining the outputs of
Modern Analytics

Provenance and Lineage in the relational setting: find tuples participating in a derivation of outputs.

✗ Potentially enormous, esp for iterative compute.

✗ Incorrect when operators are non-monotonic.
Potentially Enormous Explanations

Consider the **transitive closure**: iterated self join until fixed point.

- Every A to B path is an explanation for \((A, B)\).

A focus on completeness results in many explanations.
Incorrect for Non-Monotonic Operators

Consider counting words unique to each document.

\[
\begin{align*}
\text{doc1: } & \text{“the quick fox”} \quad \rightarrow \quad (\text{doc1, 2}) \\
\text{doc2: } & \text{“the lazy brown dog”} \quad \rightarrow \quad (\text{doc2, 3})
\end{align*}
\]

All forms of provenance produce mappings:

\[(\text{doc1, 2}) \rightarrow \text{doc1} \quad \text{and} \quad (\text{doc2, 3}) \rightarrow \text{doc2}\]

But, these inputs do not (alone) produce the outputs.

(this is known, for example in the RAMP paper)
Explaining Outputs
Explaining Outputs using differential dataflow

Imagine a language with Map + Reduce + Iteration.

(and maybe + Join + Concat + Distinct + etc.)

```racket
// transitive closure-ish
edges.iterate(|trans|
    trans.join(edges)
    .map(|(a, b, c)| (a, c))
    .concat(edges)
    .distinct()
);
```
Imagine a language with Map + Reduce + Iteration.

(and maybe + Join + Concat + Distinct + etc.)

```r
data = c(1, 2, 3)
result = reduce(data, +)  # 6
```

trans: `((a,b), +1, 0) ((b,c), +1, 0)`
Imagine a language with Map + Reduce + Iteration.

(and maybe + Join + Concat + Distinct + etc.)

```rust
// transitive closure-ish
edges.iterate(|trans|
    trans.join(edges)
      .map(|(a,b,c)| (a,c))
      .concat(edges)
      .distinct()
);
```

trans:

```
((a,b), +1, 0)
((b,c), +1, 0)
((a,c), +1, 1)
```
Imagine a language with Map + Reduce + Iteration.

(and maybe + Join + Concat + Distinct + etc.)

```rust
// transitive closure-ish
edges.iterate(|trans|
    trans.join(edges)
    .map(|(a,b,c)| (a,c))
    .concat(edges)
    .distinct()
);
```

trans:

```
(((a,b), +1, 0))
(((b,c), +1, 0))
(((a,c), +1, 1))
```
Imagine a language with Map + Reduce + Iteration.
(and maybe + Join + Concat + Distinct + etc.)

```rust
// transitive closure-ish
edges.iterate(|trans|
    trans.join(edges)
        .map(|(a, b, c)| (a, c))
        .concat(edges)
        .distinct()
);

trans:
    (((a, b), 1, (0, 0)))
    (((b, c), 1, (0, 0)))
    (((a, c), 1, (0, 1)))

(Epoch, Iteration)
```
Imagine a language with Map + Reduce + Iteration. (and maybe + Join + Concat + Distinct + etc.)

```
// transitive closure-ish
edges.iterate(!trans, |
    trans.join(edges)
    .map(|(a,b,c)| (a,c))
    .concat(edges)
    .distinct()
);
```

**trans:**
- `((a,b), +1, (0,0))`
- `((b,c), +1, (0,0))`
- `((a,c), +1, (0,1))`
- `((a,c), +1, (1,0))`
Imagine a language with Map + Reduce + Iteration.

(and maybe + Join + Concat + Distinct + etc.)

// transitive closure-ish
edges.iterate(|trans|
    trans.join(edges)
        .map(|(a,b,c)| (a,c))
    .concat(edges)
    .distinct()
);

Append-only list of diffs to the logical collection

trans:
    (((a,b), +1, (0,0)))
    (((b,c), +1, (0,0)))
    (((a,c), +1, (0,1)))
    (((a,c), +1, (1,0)))
    (((a,c), -1, (1,1)))
    ...
    ...

Explaining Outputs using differential dataflow
Explaining Outputs using differential dataflow

Append-only list of diffs to the logical collection

Track the “provenance” equivalent for these diffs.

Addresses problem #1: explanations are too big.

(partially addresses it, at least)

trans:

- $((a,b), +1, (0,0))$
- $((b,c), +1, (0,0))$
- $((a,c), +1, (0,1))$
- $((a,c), +1, (1,0))$
- $((a,c), -1, (1,1))$

...
Explaining Outputs using differential dataflow

Track the “provenance” equivalent for these diffs.

Original dataflow:
Explaining Outputs using differential dataflow

Track the “provenance” equivalent for these diffs.

Original dataflow:

Explanation dataflow:
Explaining Outputs using differential dataflow

Track the “provenance” equivalent for these diffs.

Original dataflow:

Explanation dataflow:
Explaining Outputs using differential dataflow

Track the “provenance” equivalent for these diffs using differential dataflow.

Original dataflow:

Explanation dataflow:
Explaining Outputs using differential dataflow

Track the “provenance” equivalent for these diffs using differential dataflow.

Original dataflow:

Explaination dataflow:
Explaining Outputs using differential dataflow

Track the “provenance” equivalent for these diffs using differential dataflow.

Original dataflow:

Explanation dataflow:
Explaining Outputs using differential dataflow

Track the “provenance” equivalent for these diffs using differential dataflow.

Original dataflow:

Explanation dataflow:
Explaining Outputs using differential dataflow

Track the “provenance” equivalent for these diffs using differential dataflow.

Original dataflow:

Explanation dataflow:
Explaining Outputs using differential dataflow

Track the “provenance” equivalent for these diffs using differential dataflow.

Original dataflow:

Explanation dataflow:
The **reduce** operator

\[
\text{input} \xrightarrow{((k,v1),w,t1)} \text{Reduce} \xrightarrow{((k,v2),w,t2)} \text{output}
\]

\[
\text{input} \xrightarrow{((k,v1),w,t1)} \text{Join} \xrightarrow{t1 \leq t2} \text{output}
\]
The **map** operator

... when the map function is invertible ...
The **map** operator

... when the map function is invertible ...
The **join** operator

- **Input:** ((k,v1),w,t1) and ((k,v2),w,t2)
- **Join:** ((k,(v1,v2)),w,t3)
- **Output:**
- **Query:**
- **Expl:** ((k,v1),w,t1) and ((k,v2),w,t2)
- **Join:** ((k,(v1,v2)),w,t3)
- **Output:**
The **join** operator

Note: we fake out timestamps $t_1$ and $t_2$. 

```
Map
input ((k,v1), w, t1) ((k,(v1,v2)), w, t3)
input ((k,v2), w, t2) ((k,(v1,v2)), w, t3)
expl (k,v1), w, t3 (k,(v1,v2)), w, t3
expl (k,v2), w, t3 (k,(v1,v2)), w, t3
```
The **topk** operator requires the values $v_1$ be ordered.
Explaining Outputs using differential dataflow

Track the “provenance” equivalent for these diffs using differential dataflow.

Original dataflow:

- **input** → Op → Op → output

Explanation dataflow:

- expl → Op → Op → query

**Works great for monotonic operators!**
Explaining Outputs

Original dataflow:

Explanation dataflow:
Explaining Outputs concisely

Original dataflow:

Explanation dataflow:
Explaining Outputs concisely and correctly

Original dataflow:

Explanation dataflow:
Explaining Outputs concisely and correctly

Original dataflow:

Explanation dataflow:

Test dataflow:
Explaining Outputs concisely and correctly

Original dataflow:

Explanation dataflow:

Test dataflow:
Explaining Outputs
concisely and correctly
Explaining Outputs concisely and correctly

Perhaps surprisingly:
1. This actually works!
2. It’s about 150 lines.
3. Updates in real-time.
4. Limited overhead.

Experiments:
- graph connectivity (monotonic). works great!
- stable matching (non-monotonic). works good!
Graph Connectivity
Twitter graph (1.5B edges; 42M nodes)

// label propagation in graphs
init.iterate(|values|
    values.join(edges)
    .map(|(src,(lbl,dst))| (dst,(lbl,src)))
    .concat(values)
    .topk(1, |(lbl,src)| (lbl,lbl))
)
Graph Connectivity

Twitter graph (1.5B edges; 42M nodes)
Graph Connectivity

Twitter graph (1.5B edges; 42M nodes)
Graph Connectivity

Twitter graph (1.5B edges; 42M nodes)

// label propagation in graphs
init.iterate(|values|
    values.join(edges)
        .map(|(src,(lbl,dst))| (dst,(lbl,src)))
    .concat(values)
        .topk(1, |(lbl,src)| (lbl,lbl)))
Graph Connectivity

Twitter graph (1.5B edges; 42M nodes)

with \textbf{topk} optimization

![Graph Connectivity Diagram](image)
Graph Connectivity
Twitter graph (1.5B edges; 42M nodes)
Stable Matching
LiveJournal graph (68M edges, 5M nodes)

initial.iterate(|active|
    // key records by a, order by pa.
    let props = active.map(|x| (x.a,(x.pa,x.b,x.pb)))
    .topk(1, |x| x);

    // key records by b, order by pb.
    let accpt = props.map(|x| (x.b,(x.pb,x.a,x.pa)))
    .topk(1, |x| x);

    // discard unaccepted proposals.
    active.except(props.except(accpt))
)
Stable Matching

LiveJournal graph (68M edges, 5M nodes)
Stable Matching
LiveJournal graph (68M edges, 5M nodes)

Diagram: Scatter plot showing the relationship between query time and size of explanation, with a logarithmic scale for both axes.
Stable Matching
LiveJournal graph (68M edges, 5M nodes)
Stable Matching
LiveJournal graph (68M edges, 5M nodes)
Explaining Outputs
concisely and correctly

Perhaps surprisingly:
1. This actually works!
2. It’s about 150 lines.
3. Updates in real-time.
4. Limited overhead.

Paper available at
http://e-collection.library.ethz.ch/view/eth:48721

Code is available on request; should be public soon.