Pessimistic Query Optimization: Tighter Upper Bounds for Intermediate Join Cardinalities

Walter Cai  Magdalena Balazinska  Dan Suciu

University of Washington

[walter,magda,suciu]@cs.washington.edu

April 23, 2019
Contributions

- Technique for tightening theoretically guaranteed join cardinality upper bounds.
Contributions

- Technique for tightening theoretically guaranteed join cardinality upper bounds.
- Method for enumerating practical subset of bounding formulas.
Contributions

- Technique for tightening theoretically guaranteed join cardinality upper bounds.
- Method for enumerating practical subset of bounding formulas.
- Partition budgeting strategy to control the space complexity of our sketches, and the time complexity of our bound calculation.
Contributions

- Technique for tightening theoretically guaranteed join cardinality upper bounds.
- Method for enumerating practical subset of bounding formulas.
- Partition budgeting strategy to control the space complexity of our sketches, and the time complexity of our bound calculation.
- Demonstrate practicality on challenging real world benchmark.
Query Optimization

Motivating Example

Prior Work: Cardinality Bounds

Tightened Cardinality Bounds

Optimizations

Evaluation
  - Bound Tightening
  - Runtime Improvement

Conclusion and Future Directions
Query Optimization

- Accepts queries.
- Picks “best” physical plan.
  - Could be millions of correct physical plans!
  - Conceptually, a tree with leaves as base relations.
Query Optimization

- Accepts queries.
- Picks “best” physical plan.
  - Could be millions of correct physical plans!
  - Conceptually, a tree with leaves as base relations.
Query Optimization

- Accepts queries.
- Picks “best” physical plan.
  - Could be millions of correct physical plans!
  - Conceptually, a tree with leaves as base relations.
Query Optimization

- Accepts queries.
- Picks “best” physical plan.
  - Could be millions of correct physical plans!
  - Conceptually, a tree with leaves as base relations.
Query Optimization

- Accepts queries.
- Picks “best” physical plan.
  - Could be millions of correct physical plans!
  - Conceptually, a tree with leaves as base relations.

Figure: Join tree illustrations.
Query Optimization

- Cost-Based.
  - Large parameterized summation.
  - Sum over cost of each physical operator.

Figure: Join tree illustrations.
Join Algorithms are generally binary so the DBMS will generate intermediate relations.

**Cardinality Estimation** : how large will these intermediate relations be?

**Figure**: Join tree illustrations.
Cardinality Estimation Error

- Systems rely on strong assumptions about the underlying data.
Cardinality Estimation Error

- Systems rely on strong assumptions about the underlying data.
- Assume independence of attribute value distributions across columns.
Cardinality Estimation Error

- Systems rely on strong assumptions about the underlying data.
- Assume independence of attribute value distributions across columns.
- Leads to underestimation.
  - Real world data is correlated.
  - Underestimation is risky: leads to massive blow-up from poor join orderings/algorithm choice.
Motivating Example

1. Query Optimization
2. Motivating Example
3. Prior Work: Cardinality Bounds
4. Tightened Cardinality Bounds
5. Optimizations
6. Evaluation
   - Bound Tightening
   - Runtime Improvement
7. Conclusion and Future Directions
Join Order Benchmark (JOB)

- Built on the IMDb dataset.
Join Order Benchmark (JOB)

- Built on the IMDb dataset.
  - 113 queries.
Join Order Benchmark (JOB)

- Built on the IMDb dataset.
  - 113 queries.
  - 33 unique topologies.
Join Order Benchmark (JOB)

- Built on the IMDb dataset.
  - 113 queries.
  - 33 unique topologies.
  - Skew!
Join Order Benchmark (JOB)

- Built on the IMDb dataset.
  - 113 queries.
  - 33 unique topologies.
  - Skew!
  - Correlation!
Join Order Benchmark (JOB)

- Built on the IMDb dataset.
  - 113 queries.
  - 33 unique topologies.
  - Skew!
  - Correlation!
  - Complex selection predicates!
JOB Example Query

```sql
SELECT *
FROM aka,
    cast,
    company_name,
    movie_companies,
    name,
    role,
    title
WHERE company_name.country = 'usa' AND
    role.type = 'writer' AND
    aka.person_id = name.id AND
    cast.person_id = name.id AND
    aka.person_id = cast.person_id AND
    cast.movie_id = title.id AND
    movie_companies.movie = title.id_id AND
    cast.movie_id = movie_companies.movie_id AND
    movie_companies.company_id = company_name.id AND
    cast.role_id = role.id;
```
JOB Example Query

```
SELECT *
FROM aka,
     cast,
     company_name,
     movie_companies,
     name,
     role,
     title
WHERE company_name.country = 'usa' AND
cast.movie.id = title.id AND
cast.role.id = role.id
```

Figure: Join Graph.
SELECT *
FROM aka, cast, company_name, movie_companies, name, role, title
WHERE company_name.country = 'usa' AND role.type = 'writer' AND aka.person_id = name.id AND cast.person_id = name.id AND aka.person_id = cast.person_id AND cast.movie_id = title.id AND movie_companies.movie = title.id AND cast.movie_id = movie_companies.movie_id AND movie_companies.company_id = company_name.id AND cast.role_id = role.id;

Figure: Join Graph.
JOB Example Query

```sql
SELECT *
FROM aka, cast, company_name, movie_companies, name, role, title
WHERE company_name.country = 'usa' AND role.type = 'writer' AND aka.person_id = name.id AND cast.person_id = name.id AND aka.person_id = cast.person_id AND cast.movie_id = title.id AND movie_companies.movie = title.id AND cast.movie_id = movie_companies.movie_id AND movie_companies.company_id = company_name.id AND cast.role_id = role.id;
```

Figure: Join Graph.
Figure: Join Graph.

```
SELECT * 
FROM aka, cast, company_name, movie_companies, name, role, title 
WHERE company_name.country = 'usa' AND 
    role.type = 'writer' AND 
    aka.person_id = name.id AND 
    cast.person_id = name.id AND 
   aka.person_id = cast.person_id AND 
    cast.movie_id = title.id AND 
    movie_companies.movie = title.id_id AND 
    cast.movie_id = movie_companies.movie_id AND 
    movie_companies.company_id = company_name.id AND 
    cast.role_id = role.id;
```
JOB Example Query

```sql
SELECT *
FROM aka, cast, company_name, movie_companies, name, role, title
WHERE company_name.country = 'usa' AND role.type = 'writer' AND aka.person_id = name.id AND cast.person_id = name.id AND aka.person_id = cast.person_id AND cast.movie_id = title.id AND movie_companies.movie = title.id_id AND cast.movie_id = movie_companies.movie_id AND movie_companies.company_id = company_name.id AND cast.role_id = role.id;
```

Figure: Join Graph.
SELECT *
FROM aka, cast, company_name, movie_companies, name, role, title
WHERE company_name.country = 'usa' AND role.type = 'writer' AND aka.person_id = name.id AND cast.person_id = name.id AND aka.person_id = cast.person_id AND cast.movie_id = title.id AND movie_companies.movie = title.id_id AND cast.movie_id = movie_companies.movie_id AND movie_companies.company_id = company_name.id AND cast.role_id = role.id;
JOB Example Query

Figure: Join Graph.

```
SELECT *
FROM aka, cast, company_name, movie_companies, name, role, title
WHERE company_name.country = 'usa' AND role.type = 'writer' AND aka.person_id = name.id AND cast.person_id = name.id AND aka.person_id = cast.person_id AND cast.movie_id = title.id AND movie_companies.movie = title.id_id AND cast.movie_id = movie_companies.movie_id AND movie_companies.company_id = company_name.id AND cast.role_id = role.id;
```
JOB Example Query

```
SELECT *
FROM aka, cast, company_name, movie_companies
WHERE aka.person_id = cast.person_id
AND cast.movie_id = movie_companies.movie_id
AND movie_companies.company_id = company_name.id;
```

Figure: Join Graph.
**JOB Example Query**

**SELECT**

*        
**FROM**

aka,  
    cast,  
    company_name,  
    movie_companies

**WHERE**

aka.person_id = cast.person_id AND  
    cast.movie_id = movie_companies.movie_id AND  
    movie_companies.company_id = company_name.id;

\[
Q(x, y, z, w) : -  
aka(x, y),  
    cast(y, z),  
    movie_companies(z, w),  
    company_name(w)
\]

**Figure:** Join Graph.
Motivating Example

Worst Case Scenario

\[ Q(x, y, z, w) := aka(x, y), \text{cast}(y, z), \text{movie\_companies}(z, w), \text{company\_name}(w) \]
Worst Case Scenario

\[ Q(x, y, z, w) :\neg \text{aka}(x, y), \text{cast}(y, z), \text{movie} \_ \text{companies}(z, w), \text{company} \_ \text{name}(w) \]
A Better Plan

\[ Q(x, y, z, w) :\text{\texttt{aka}}(x, y), \text{\texttt{cast}}(y, z), \text{\texttt{movie\_companies}}(z, w), \text{\texttt{company\_name}}(w) \]
Prior Work: Cardinality Bounds

Query Optimization

Motivating Example

Prior Work: Cardinality Bounds

Tightened Cardinality Bounds

Optimizations

Evaluation
  - Bound Tightening
  - Runtime Improvement

Conclusion and Future Directions
Review: Entropy

Take random variable $X$:

$$h(X) = - \sum_a P(X = a) \cdot \log(P(X = a))$$

Multiple variables:

$$h(X, Y) = - \sum_{a,b} P(X = a, Y = b) \cdot \log(P(X = a, Y = b))$$

Conditional Entropy:

$$h(X|Y) = - \sum_{a,b} P(X = a, Y = b) \cdot \log\left(\frac{P(X = a, Y = b)}{P(Y = b)}\right)$$
Prior Work: Cardinality Bounds

Review: Entropy

Let $X$ be uniformly distributed on the space $\{a_1, a_2, \ldots, a_n\}$.

\[
h(X) = - \sum_{i=1}^{n} P(X = a_i) \cdot \log(P(X = a_i))
\]

\[
= - \sum_{i=1}^{n} \frac{1}{n} \cdot \log \left( \frac{1}{n} \right)
\]

\[
= -n \frac{1}{n} \cdot \log \left( \frac{1}{n} \right)
\]

\[
= \log \left( n \right)
\]
Connection to Entropy

\[ Q(x, y, z, w) :\text{aka}(x, y), \text{cast}(y, z), \text{movie\_companies}(z, w), \text{company\_name}(w) \]
Connection to Entropy

\[ Q(x, y, z, w) :\text{\texttt{aka}}(x, y), \text{\texttt{cast}}(y, z), \text{\texttt{movie}}\textunderscore\text{\texttt{companies}}(z, w), \text{\texttt{company}}\textunderscore\text{\texttt{name}}(w) \]

▶ Create a random variable for each of the attributes present in the query.

\[
\begin{align*}
x & \rightarrow X, \quad y \rightarrow Y, \quad z \rightarrow Z, \quad w \rightarrow W
\end{align*}
\]
Connection to Entropy

\[ Q(x, y, z, w) :\sim \text{aka}(x, y), \text{cast}(y, z), \text{movie\_companies}(z, w), \text{company\_name}(w) \]

- Create a random variable for each of the attributes present in the query.

\[ x \rightarrow X, \quad y \rightarrow Y, \quad z \rightarrow Z, \quad w \rightarrow W \]

- Let \((X, Y, Z, W)\) be uniformly distributed over all tuples in the true output of \(Q\).

\[ \left| Q(x, y, z, w) \right| = \exp\left( h(X, Y, Z, W) \right) \]
Entropic Bounds

\[ |Q(x, y, z, w)| = \exp\left( h(X, Y, Z, W) \right) \]

- Suffices to bound \( h(X, Y, Z, W) \).
Entropic Bounds

\[ |Q(x, y, z, w)| = \exp\left( h(X, Y, Z, W) \right) \]

- Suffices to bound \( h(X, Y, Z, W) \).
- There are plenty of entropic bounds to choose from!
Entropic Bounds

\[
\begin{align*}
&h(X, Y, Z, W) \leq \\
&1 \ h(X, Y) + h(Z|Y) + h(W|Z) \\
&2 \ h(X, Y) + h(Z|Y) + h(W) \\
&3 \ h(X, Y) + h(Z, W) \\
&4 \ h(X, Y) + h(Z|W) + h(W) \\
&5 \ h(X|Y) + h(Y, Z) + h(W|Z) \\
&6 \ h(X|Y) + h(Y, Z) + h(W) \\
&7 \ h(X|Y) + h(Y|Z) + h(Z, W) \\
&8 \ h(X|Y) + h(Y|Z) + h(Z|W) + h(Z)
\end{align*}
\]

(only a subset of all entropic bounding formulas)
Entropic Bounds

\[
\begin{align*}
\text{1} & \quad h(X, Y) + h(Z|Y) + h(W|Z) \\
\text{2} & \quad h(X, Y) + h(Z|Y) + h(W) \\
\text{3} & \quad h(X, Y) + h(Z, W) \\
\text{4} & \quad h(X, Y) + h(Z|W) + h(W) \\
\text{5} & \quad h(X|Y) + h(Y, Z) + h(W|Z) \\
\text{6} & \quad h(X|Y) + h(Y, Z) + h(W) \\
\text{7} & \quad h(X|Y) + h(Y|Z) + h(Z, W) \\
\text{8} & \quad h(X|Y) + h(Y|Z) + h(Z|W) + h(Z)
\end{align*}
\]

(only a subset of all entropic bounding formulas)
Entropic Bounds

\[ |Q(x, y, z, w)| = \exp(h(X, Y, Z, W)) \leq \exp(h(X|Y) + h(Y, Z) + h(W|Z)) \]

\[
\begin{align*}
    h(X|Y) & \leq \log d_{aka}^y \\
    h(Y, Z) & \leq \log c_{cast} \\
    h(W|Z) & \leq \log d_{movie\_companies}^z
\end{align*}
\]

\[ d_{aka}^y = \text{"Max Degree"} \]
\[ = \text{Count of most common } y \text{ attribute value in } aka\_name. \]

\[ c_{cast} = \text{"Count"} \]
\[ = \text{Count of entire cast\_info relation.} \]
|Q(x, y, z, w)| = \exp(h(X, Y, Z, W)) \\
\leq \exp(h(X|Y) + h(Y, Z) + h(W|Z)) \\
\leq \log d^y_{aka} + \log c_{cast} + \log d^z_{movie.companies} \\
\leq d^y_{aka} \cdot c_{cast} \cdot d^z_{movie.companies}
Prior Work: Cardinality Bounds

Bound Formula Generation

\[ Q(x, y, z, w) :\leftarrow \text{aka}(x, y), \text{cast}(y, z), \text{movie\_companies}(z, w), \text{company\_name}(w) \]

<table>
<thead>
<tr>
<th>(x)</th>
<th>(y)</th>
<th>(z)</th>
<th>(w)</th>
<th>entropic formula</th>
<th>bound formula</th>
</tr>
</thead>
<tbody>
<tr>
<td>aka</td>
<td>aka</td>
<td>cast</td>
<td>mc</td>
<td>(h(X, Y) + h(Z</td>
<td>Y) + h(W</td>
</tr>
<tr>
<td>aka</td>
<td>aka</td>
<td>cast</td>
<td>cn</td>
<td>(h(X, Y) + h(Z</td>
<td>Y) + h(W))</td>
</tr>
<tr>
<td>aka</td>
<td>aka</td>
<td>mc</td>
<td>mc</td>
<td>(h(X, Y) + h(Z, W))</td>
<td>(c_{\text{aka}} \cdot c_{\text{mc}})</td>
</tr>
<tr>
<td>aka</td>
<td>aka</td>
<td>mc</td>
<td>cn</td>
<td>(h(X, Y) + h(Z</td>
<td>W) + h(W))</td>
</tr>
<tr>
<td>aka</td>
<td>cast</td>
<td>cast</td>
<td>mc</td>
<td>(h(X</td>
<td>Y) + h(Y, Z) + h(W</td>
</tr>
<tr>
<td>aka</td>
<td>cast</td>
<td>cast</td>
<td>cn</td>
<td>(h(X</td>
<td>Y) + h(Y, Z) + h(W))</td>
</tr>
<tr>
<td>aka</td>
<td>cast</td>
<td>mc</td>
<td>mc</td>
<td>(h(X</td>
<td>Y) + h(Y</td>
</tr>
<tr>
<td>aka</td>
<td>cast</td>
<td>mc</td>
<td>cn</td>
<td>(h(X</td>
<td>Y) + h(Y</td>
</tr>
</tbody>
</table>
Bound Formula Generation

\[ Q(x, y, z, w) :\neg \text{aka}(x, y), \text{cast}(y, z), \text{movie\_companies}(z, w), \text{company\_name}(w) \]

<table>
<thead>
<tr>
<th>x</th>
<th>y</th>
<th>z</th>
<th>w</th>
<th>entropic formula</th>
<th>bound formula</th>
</tr>
</thead>
<tbody>
<tr>
<td>aka</td>
<td>aka</td>
<td>cast</td>
<td>mc</td>
<td>(h(X, Y) + h(Z</td>
<td>Y) + h(W</td>
</tr>
<tr>
<td>aka</td>
<td>aka</td>
<td>cast</td>
<td>cn</td>
<td>(h(X, Y) + h(Z</td>
<td>Y) + h(W))</td>
</tr>
<tr>
<td>aka</td>
<td>aka</td>
<td>mc</td>
<td>mc</td>
<td>(h(X, Y) + h(Z, W))</td>
<td>(c_{\text{aka}} \cdot c_{\text{mc}})</td>
</tr>
<tr>
<td>aka</td>
<td>aka</td>
<td>mc</td>
<td>cn</td>
<td>(h(X, Y) + h(Z</td>
<td>W) + h(W))</td>
</tr>
<tr>
<td>aka</td>
<td>cast</td>
<td>cast</td>
<td>mc</td>
<td>(h(X</td>
<td>Y) + h(Y, Z) + h(W</td>
</tr>
<tr>
<td>aka</td>
<td>cast</td>
<td>cast</td>
<td>cn</td>
<td>(h(X</td>
<td>Y) + h(Y, Z) + h(W))</td>
</tr>
<tr>
<td>aka</td>
<td>cast</td>
<td>mc</td>
<td>mc</td>
<td>(h(X</td>
<td>Y) + h(Y</td>
</tr>
<tr>
<td>aka</td>
<td>cast</td>
<td>mc</td>
<td>cn</td>
<td>(h(X</td>
<td>Y) + h(Y</td>
</tr>
</tbody>
</table>
Bound Formula Generation

\[ Q(x, y, z, w) :\neg \text{aka}(x, y), \text{cast}(y, z), \text{movie\_companies}(z, w), \text{company\_name}(w) \]

<table>
<thead>
<tr>
<th>x</th>
<th>y</th>
<th>z</th>
<th>w</th>
<th>entropic formula</th>
<th>bound formula</th>
</tr>
</thead>
<tbody>
<tr>
<td>aka</td>
<td>aka</td>
<td>cast</td>
<td>mc</td>
<td>( h(X, Y) + h(Z</td>
<td>Y) + h(W</td>
</tr>
<tr>
<td>aka</td>
<td>aka</td>
<td>cast</td>
<td>cn</td>
<td>( h(X, Y) + h(Z</td>
<td>Y) + h(W) )</td>
</tr>
<tr>
<td>aka</td>
<td>aka</td>
<td>mc</td>
<td>mc</td>
<td>( h(X, Y) + h(Z, W) )</td>
<td>( c_{\text{aka}} \cdot c_{\text{mc}} )</td>
</tr>
<tr>
<td>aka</td>
<td>aka</td>
<td>mc</td>
<td>cn</td>
<td>( h(X, Y) + h(Z</td>
<td>W) + h(W) )</td>
</tr>
<tr>
<td>aka</td>
<td>cast</td>
<td>cast</td>
<td>mc</td>
<td>( h(X</td>
<td>Y) + h(Y, Z) + h(W</td>
</tr>
<tr>
<td>aka</td>
<td>cast</td>
<td>cast</td>
<td>cn</td>
<td>( h(X</td>
<td>Y) + h(Y, Z) + h(W) )</td>
</tr>
<tr>
<td>aka</td>
<td>cast</td>
<td>mc</td>
<td>mc</td>
<td>( h(X</td>
<td>Y) + h(Y</td>
</tr>
<tr>
<td>aka</td>
<td>cast</td>
<td>mc</td>
<td>cn</td>
<td>( h(X</td>
<td>Y) + h(Y</td>
</tr>
</tbody>
</table>
## Bound Formula Generation

$$Q(x, y, z, w) :\neg \textit{aka}(x, y), \textit{cast}(y, z), \textit{movie\_companies}(z, w), \textit{company\_name}(w)$$

<table>
<thead>
<tr>
<th>x</th>
<th>y</th>
<th>z</th>
<th>w</th>
<th>entropic formula</th>
<th>bound formula</th>
</tr>
</thead>
<tbody>
<tr>
<td>aka</td>
<td>aka</td>
<td>cast</td>
<td>mc</td>
<td>$h(X, Y) + h(Z</td>
<td>Y) + h(W</td>
</tr>
<tr>
<td>aka</td>
<td>aka</td>
<td>cast</td>
<td>cn</td>
<td>$h(X, Y) + h(Z</td>
<td>Y) + h(W)$</td>
</tr>
<tr>
<td>aka</td>
<td>aka</td>
<td>mc</td>
<td>mc</td>
<td>$h(X, Y) + h(Z, W)$</td>
<td>$c_{aka} \cdot c_{\text{mc}}$</td>
</tr>
<tr>
<td>aka</td>
<td>aka</td>
<td>mc</td>
<td>cn</td>
<td>$h(X, Y) + h(Z</td>
<td>W) + h(W)$</td>
</tr>
<tr>
<td>aka</td>
<td>cast</td>
<td>cast</td>
<td>mc</td>
<td>$h(X</td>
<td>Y) + h(Y, Z) + h(W</td>
</tr>
<tr>
<td>aka</td>
<td>cast</td>
<td>cast</td>
<td>cn</td>
<td>$h(X</td>
<td>Y) + h(Y, Z) + h(W)$</td>
</tr>
<tr>
<td>aka</td>
<td>cast</td>
<td>mc</td>
<td>mc</td>
<td>$h(X</td>
<td>Y) + h(Y</td>
</tr>
<tr>
<td>aka</td>
<td>cast</td>
<td>mc</td>
<td>cn</td>
<td>$h(X</td>
<td>Y) + h(Y</td>
</tr>
</tbody>
</table>
Entropic Bounds

Neat! But is it useful?

- Short answer: No. (Not yet, anyway)
Prior Work: Cardinality Bounds

Entropic Bounds

Neat! But is it useful?
- Short answer: No. (Not yet, anyway)
  - Bounds are still far too loose (overestimation).
Entropic Bounds

Neat! But is it useful?

- Short answer: No. (Not yet, anyway)
  - Bounds are still far too loose (overestimation).
  - Need to tighten the bounds.
Entropic Bounds

Neat! But is it useful?

- Short answer: No. (Not yet, anyway)
  - Bounds are still far too loose (overestimation).
  - Need to tighten the bounds.

- How to tighten? Partitioning.
Tightened Cardinality Bounds

1. Query Optimization
2. Motivating Example
3. Prior Work: Cardinality Bounds
4. Tightened Cardinality Bounds
5. Optimizations
6. Evaluation
   - Bound Tightening
   - Runtime Improvement
7. Conclusion and Future Directions
\[ Q(x, y, z, w) :\text{\texttt{aka}}(x, y), \text{\texttt{cast}}(y, z), \text{\texttt{movie\_companies}}(z, w), \text{\texttt{company\_name}}(w) \]
Tightened Cardinality Bounds

\[ Q(x, y, z, w) : \neg \text{aka}(x, y), \text{cast}(y, z), \text{movie}\_\text{companies}(z, w), \text{company}\_\text{name}(w) \]

<table>
<thead>
<tr>
<th>aka</th>
<th>cast</th>
<th>movie_companies</th>
<th>company_name</th>
</tr>
</thead>
<tbody>
<tr>
<td>aka[0,0]</td>
<td>cast[0,0]</td>
<td>mc[0,0]</td>
<td>cn[0]</td>
</tr>
<tr>
<td>aka[1,0]</td>
<td>cast[0,1]</td>
<td>mc[0,1]</td>
<td>cn[1]</td>
</tr>
<tr>
<td>aka[0,1]</td>
<td>cast[1,0]</td>
<td>mc[0,0]</td>
<td></td>
</tr>
<tr>
<td>aka[1,1]</td>
<td>cast[1,1]</td>
<td>mc[1,1]</td>
<td></td>
</tr>
</tbody>
</table>

Hash the values of each tuple and bucketize on the hash values.

\[ \text{aka}[1, 0] = \left\{ t \in \text{cast} \left| \text{hash}(t[y]) = 1 \land \text{hash}(t[z]) = 0 \right. \right\} \]
$Q(x, y, z, w) : \text{- aka}(x, y), \text{cast}(y, z), \text{movie\_companies}(z, w), \text{company\_name}(w)$

- Pick a hash value for each attribute in the query:
  \[x, y, z, w \rightarrow [0, 1, 0, 1]\]

- The matching buckets from each relation is the partition $D[0, 1, 0, 1]$. 
\[ Q(x, y, z, w) :- \textit{aka}(x, y), \textit{cast}(y, z), \textit{movie.companies}(z, w), \textit{company.name}(w) \]

- \( Q(D) \): query evaluated on database \( D \).
- \( Q(D[J]) \): query evaluated on partition \( D[J] \).
Tightened Cardinality Bounds

\[ Q(x, y, z, w) := \text{aka}(x, y), \text{cast}(y, z), \text{movie\_companies}(z, w), \text{company\_name}(w) \]

- Bound each partition \( D[J] \).
- Sum will be a bound on the full database \( D \).

\[ Q(D) = \bigcup J Q(D[J]) \]
\[ |Q(D)| \leq \sum J \text{bound}(Q(D[J])) \]
Partition Bounding

\[ |Q(D)| \leq \sum_{J \in \{0,1\}^4} \min \begin{cases} 
  c_{aka}[J] \cdot d^y_{cast}[J] \cdot d^z_{mc}[J] \\
  c_{aka}[J] \cdot d^y_{cast}[J] \cdot c_{cn}[J] \\
  c_{aka}[J] \cdot c_{mc}[J] \\
  c_{aka}[J] \cdot d^w_{mc}[J] \cdot c_{cn}[J] \\
  d^y_{aka}[J] \cdot c_{cast}[J] \cdot d^z_{mc}[J] \\
  d^y_{aka}[J] \cdot c_{cast}[J] \cdot c_{cn}[J] \\
  d^y_{aka}[J] \cdot d^z_{cast}[J] \cdot c_{mc}[J] \\
  d^y_{aka}[J] \cdot d^z_{cast}[J] \cdot d^w_{mc}[J] \cdot c_{cn}[J]
\end{cases} \]
The Bound Sketch

- One bound sketch per table.
The Bound Sketch

- One bound sketch per table.
- Need count and degree statistics.
One bound sketch per table.

Need count and degree statistics.

Some calculated offline, some at runtime.
The Bound Sketch

<table>
<thead>
<tr>
<th>aka</th>
<th>cast</th>
<th>movie_companies</th>
<th>company_name</th>
</tr>
</thead>
<tbody>
<tr>
<td>aka[0,0]</td>
<td>cast[0,0]</td>
<td>mc[0,0]</td>
<td>cn[0]</td>
</tr>
<tr>
<td>aka[0,1]</td>
<td>cast[0,1]</td>
<td>mc[0,1]</td>
<td>cn[1]</td>
</tr>
<tr>
<td>aka[1,0]</td>
<td>cast[1,0]</td>
<td>mc[1,0]</td>
<td></td>
</tr>
<tr>
<td>aka[1,1]</td>
<td>cast[1,1]</td>
<td>mc[1,1]</td>
<td></td>
</tr>
</tbody>
</table>

- One bound sketch per table.
- Need count and degree statistics.
- Some calculated offline, some at runtime.
- Like a richer randomized histogram.
### The Bound Sketch

<table>
<thead>
<tr>
<th>aka</th>
<th>cast</th>
<th>movie.companies</th>
<th>company_name</th>
</tr>
</thead>
<tbody>
<tr>
<td>aka[0,0]</td>
<td>cast[0,0]</td>
<td>mc[0,0]</td>
<td>cn[0]</td>
</tr>
<tr>
<td>aka[0,1]</td>
<td>cast[0,1]</td>
<td>mc[0,1]</td>
<td>cn[1]</td>
</tr>
<tr>
<td>aka[1,0]</td>
<td>cast[1,0]</td>
<td>mc[1,0]</td>
<td></td>
</tr>
<tr>
<td>aka[1,1]</td>
<td>cast[1,1]</td>
<td>mc[1,1]</td>
<td></td>
</tr>
</tbody>
</table>

- One bound sketch per table.
- Need count and degree statistics.
- Some calculated offline, some at runtime.
- Like a richer randomized histogram.
- Restriction: this method is only suitable for equijoins.
Exponential Growth

- Sketch size (number of buckets) exponential in hash size.
  - Exponent is number of attributes in relation.
Exponential Growth

- Sketch size (number of buckets) exponential in hash size.
  - Exponent is number of attributes in relation.
- Number of elements to sum up exponential in hash size.
  - Exponent is number of attributes in entire query.
Exponential Growth

- Sketch size (number of buckets) exponential in hash size.
  - Exponent is number of attributes in relation.
- Number of elements to sum up exponential in hash size.
  - Exponent is number of attributes in entire query.
- Non-monotonic bound behavior
Tuning Bucket Allocation

- As the number of buckets increases, we get more information, and bounds tighten, right?
As the number of buckets increases, we get more information, and bounds tighten, right?

When exclusively partitioning unconditionally covered attributes: yes.
Tuning Bucket Allocation

- As the number of buckets increases, we get more information, and bounds tighten, right?
- When exclusively partitioning unconditionally covered attributes: yes.
- When also partitioning conditionally covered attributes: not necessarily.
Tuning Bucket Allocation

- As the number of buckets increases, we get more information, and bounds tighten, right?
- When exclusively partitioning unconditionally covered attributes: yes.
- When also partitioning conditionally covered attributes: not necessarily.
  - Non-monotonic tradeoff space.
Example of Non-monotonic Behavior

$$Q(x, y, z, w) :- R(z, y), S(y, z), T(z, w)$$
Example of Non-monotonic Behavior

<table>
<thead>
<tr>
<th></th>
<th>x</th>
<th>y</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>y</th>
<th>z</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>3</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>z</th>
<th>w</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>x</th>
<th>y</th>
<th>z</th>
<th>w</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

\[ \bigg| Q(x, y, z, w) \bigg| \leq \min \begin{cases} c_R \cdot d_y \cdot c_S \cdot d_z \cdot c_T \cdot d_y \cdot R(0) \cdot d_y \cdot S(0) \cdot d_y \cdot T(0) \end{cases} = 4 \cdot 1 \cdot 1 = 4 \]
Example of Non-monotonic Behavior

\[
\begin{array}{cccc}
 x & y & y & z & z & w \\
 0 & 0 & 0 & 0 & 0 & 0 \\
 0 & 1 & 1 & 1 & 1 & 1 \\
 1 & 0 & 2 & 2 & 2 & 2 \\
 1 & 1 & 3 & 3 & 3 & 3 \\
\end{array}
\]

\[
\begin{array}{cccc}
 x & y & z & w \\
 0 & 0 & 0 & 0 \\
 0 & 1 & 0 & 0 \\
 1 & 0 & 0 & 0 \\
 1 & 1 & 0 & 0 \\
\end{array}
\]

\[
|Q(x, y, z, w)| \leq \min \left\{ \begin{array}{l}
 c_R \cdot d^y_R \cdot d^z_T \\
 d^y_R \cdot c_S \cdot d^z_T \\
 d^y_R \cdot d^z_S \cdot c_T \\
 c_R \cdot c_T \\
\end{array} \right\}
\]
**Example of Non-monotonic Behavior**

\[
|Q(x, y, z, w)| \leq \min \begin{cases} 
 c_R \cdot d^y_S \cdot d^z_T \\
 d^y_R \cdot c_S \cdot d^z_T \\
 d^y_R \cdot d^z_T \cdot c_T \\
 c_R \cdot c_T 
\end{cases}
\]

\[
c_R^{(0)} \cdot d^y_{S(0,0)} \cdot d^z_{T(0)} = 4 \cdot 1 \cdot 1 = 4
\]
Define hash function $h(u_i) = i \% 2$.

$\begin{align*}
    h(0) &= h(2) = 0 \\
    h(1) &= h(3) = 1
\end{align*}$
Partitioned Relations

hash(y), hash(z) = …
Tightened Cardinality Bounds

\[
\sum_{i,j \in \{0,1\}} \min \left\{ c_R(i) \cdot d_S(i,j) \cdot d_T(j), d_R(i) \cdot c_S(i,j) \cdot d_T(j), d_R(i) \cdot d_S(i,j) \cdot c_T(j), c_R(i) \cdot c_T(j) \right\}
\]
Tightened Cardinality Bounds

\[
\begin{align*}
\sum_{i,j \in \{0,1\}} \min \begin{cases} 
    c_R(i) \cdot d^y_{S(i,j)} \cdot d^z_T(j) \\
    d^{y\prime}_{R(i)} \cdot c_{S(i,j)} \cdot d^z_T(j) \\
    d^{y\prime}_{R(i)} \cdot d^z_{S(i,j)} \cdot c_T(j) \\
    c_R(i) \cdot c_T(j)
\end{cases} &= \sum_{i,j \in \{0,1\}} \min \begin{cases} 
    2 \cdot 1 \cdot 1 \\
    2 \cdot 1 \cdot 1 \\
    2 \cdot 1 \cdot 2 \\
    2 \cdot 2
\end{cases}
\end{align*}
\]
Tightened Cardinality Bounds

\[ \begin{array}{c|c|c} \hline 0,0 & 0 & 0 \\ \hline 1 & 0 \\ \hline \end{array} \quad \times \quad \begin{array}{c|c|c} \hline 0 & 0 \\ \hline 2 & 2 \\ \hline \end{array} = \begin{array}{c|c|c} \hline 0 & 0 & 0 \\ \hline 1 & 0 & 0 \\ \hline \end{array} \]

\[ \begin{array}{c|c|c} \hline 0,1 & 0 & 0 \\ \hline 1 & 0 \\ \hline \end{array} \quad \times \quad \begin{array}{c|c|c} \hline 2 & 1 \\ \hline 1 & 1 \\ \hline \end{array} \quad \times \quad \begin{array}{c|c|c} \hline 1 & 1 \\ \hline 3 & 3 \\ \hline \end{array} = \emptyset \]

\[ \begin{array}{c|c|c} \hline 1,0 & 0 & 1 \\ \hline 1 & 1 \\ \hline \end{array} \quad \times \quad \begin{array}{c|c|c} \hline 1 & 0 \\ \hline 2 & 2 \\ \hline \end{array} = \begin{array}{c|c|c} \hline 0 & 1 & 0 \\ \hline 1 & 1 & 0 \\ \hline \end{array} \]

\[ \begin{array}{c|c|c} \hline 1,1 & 0 & 1 \\ \hline 1 & 1 \\ \hline \end{array} \quad \times \quad \begin{array}{c|c|c} \hline 3 & 1 \\ \hline 1 & 1 \\ \hline \end{array} = \emptyset \]

\[
\sum_{i,j \in \{0,1\}} \min \left\{ c_{R(i)} \cdot d_{S(i,j)}^y \cdot d_{T(j)}^z, d_{R(i)}^y \cdot c_{S(i,j)} \cdot d_{T(j)}^z, d_{R(i)}^y \cdot d_{S(i,j)}^z \cdot c_{T(j)}, c_{R(i)} \cdot c_{T(j)} \right\} = \sum_{i,j \in \{0,1\}} \min \left\{ 2 \cdot 1 \cdot 1, 2 \cdot 1 \cdot 1, 2 \cdot 1 \cdot 2, 2 \cdot 2 \right\} = \sum_{i,j \in \{0,1\}} 2 = 8
\]
Non-Linearity of Degree Statistic

- Count is linear with respect to disjoint union!
  \[ \text{count}(A) + \text{count}(B) = \text{count}(A \cup B) \]

- Degree is not...
  \[ \text{degree}(A) + \text{degree}(B) \geq \text{degree}(A \cup B) \]
\[Q(x, y, z, w) : \text{aka}(x, y), \text{cast}(y, z), \text{movie}\_\text{companies}(z, w), \text{company}\_\text{name}(w)\]
\[ Q(x, y, z, w) :\sim aka(x, y), \text{cast}(y, z), \text{movie.companies}(z, w), \text{company.name}(w) \]

\[ c_{aka} \cdot d_{\text{cast}}^y \cdot d_{\text{mc}}^z \]
\[ Q(x, y, z, w) \leftarrow aka(x, y), cast(y, z), movie\_companies(z, w), company\_name(w) \]

\[ c_{aka} \cdot d_{cast}^y \cdot d_{mc}^z \]

\begin{align*}
\text{aka} & :\{aka[0], aka[1], aka[2], aka[3]\} \\
\text{cast} & :\{cast[0], cast[1], cast[2], cast[3]\} \\
\text{movie\_companies} & :\{mc\} \\
\text{company\_name} & :\{cn\} 
\end{align*}
\[ Q(x, y, z, w) := \text{aka}(x, y), \text{cast}(y, z), \text{movie\_companies}(z, w), \text{company\_name}(w) \]

\[ d^y_{\text{aka}} \cdot c_{\text{cast}} \cdot d^z_{\text{mc}} \]
$Q(x, y, z, w) :\neg\aka(x, y), \cast(y, z), \movie_companies(z, w), \company_name(w)$

$$d^y_{\aka} \cdot c_{\cast} \cdot d^z_{mc}$$
Bound Calculation

Calculate the minimal bound over all entropic bounds.

\[ |Q(D)| \leq \min_{b \in \text{bounding formulas}} \left( \sum_{J \in \text{partition indexes}} b(Q(D[J])) \right) \]
1. Query Optimization

2. Motivating Example

3. Prior Work: Cardinality Bounds

4. Tightened Cardinality Bounds

5. Optimizations

6. Evaluation
   - Bound Tightening
   - Runtime Improvement

7. Conclusion and Future Directions
Filter Predicate Analysis

```
SELECT * 
FROM aka, cast, company_name, movie_companies, name, role, title 
WHERE 
  company_name.country = 'usa' AND 
  role.type = 'writer' AND 
  aka.person_id = name.id AND 
  cast.person_id = name.id AND 
  aka.person_id = cast.person_id AND 
  cast.movie_id = title.id AND 
  movie_companies.movie = title.id AND 
  movie_companies.company_id = company_name.id AND 
  cast.role_id = role.id;
```
Filter Predicate Analysis

```
SELECT *
FROM aka,
cast,
company_name,
movie_companies,
name,
role,
title
WHERE
  company.name.country = 'usa' AND
  role.type = 'writer' AND
  aka.person_id = name.id AND
  cast.person_id = name.id AND
  aka.person_id = cast.person_id AND
  cast.movie_id = title.id AND
  movie.companies.movie = title.id AND
  cast.movie_id = movie.companies.movie AND
  movie.companies.company_id = company.name.id AND
  cast.role_id = role.id;
```
Filter Predicate Analysis

```
SELECT * 
FROM aka, cast, company_name, movie_companies, name, role, title
WHERE company_name.country = 'usa' AND 
  role.type = 'writer' AND 
  aka.person_id = name.id AND 
  cast.person_id = name.id AND 
  aka.person_id = cast.person_id AND 
  cast.movie_id = title.id AND 
  movie_companies.movie = title.id AND 
  cast.movie_id = movie_companies.movie AND 
  movie_companies.company_id = company_name.id AND 
  cast.role_id = role.id;
```
Optimizations

Filter Predicate Analysis

\[ \sigma_{\text{role} \cdot \text{id} = 4} (\text{cast}) \]

\[ \sigma_{\text{country} = 'usa'} (\text{company} \cdot \text{name}) \]

\[
\begin{align*}
\text{SELECT} \\
\quad * \\
\text{FROM} \\
\quad \text{aka,} \\
\quad \text{cast,} \\
\quad \text{company} \cdot \text{name,} \\
\quad \text{movie} \cdot \text{companies,} \\
\quad \text{name,} \\
\quad \text{role,} \\
\quad \text{title} \\
\text{WHERE} \\
\quad \text{company} \cdot \text{name} \cdot \text{country} = 'usa' \text{ AND} \\
\quad \text{role} \cdot \text{type} = 'writer' \text{ AND} \\
\quad \text{aka} \cdot \text{person} \cdot \text{id} = \text{name} \cdot \text{id} \text{ AND} \\
\quad \text{cast} \cdot \text{person} \cdot \text{id} = \text{name} \cdot \text{id} \text{ AND} \\
\quad \text{aka} \cdot \text{person} \cdot \text{id} = \text{cast} \cdot \text{person} \cdot \text{id} \text{ AND} \\
\quad \text{cast} \cdot \text{movie} \cdot \text{id} = \text{title} \cdot \text{id} \text{ AND} \\
\quad \text{movie} \cdot \text{companies} \cdot \text{movie} = \text{title} \cdot \text{id} \cdot \text{id} \text{ AND} \\
\quad \text{cast} \cdot \text{movie} \cdot \text{id} = \text{movie} \cdot \text{companies} \cdot \text{movie} \cdot \text{id} \text{ AND} \\
\quad \text{movie} \cdot \text{companies} \cdot \text{company} \cdot \text{id} = \text{company} \cdot \text{name} \cdot \text{id} \text{ AND} \\
\quad \text{cast} \cdot \text{role} \cdot \text{id} = \text{role} \cdot \text{id};
\end{align*}
\]
Filter Propagation

Figure: Original hypergraph representation.
Filter Propagation

**Figure:** Original hypergraph representation.

**Figure:** Hypergraph after selection propagation and elimination.
Table Scans During Optimization

Analysis of selection predicates can lead to:

- Full propagation.
- Highly selective predicate: yields fewer tuples than the hash size.
- Scans on predicate relation and (most likely) on foreign key relation.
- Updated the bound sketch.
- Selective predicate but more tuples than hash size.
- Scan on predicate relation.
- Defaulting to unmodified bound sketch.
- Non selective predicate.
- Early exit during scan on predicate relation.
Table Scans During Optimization

Analysis of selection predicates can lead to:

- Full propagation.
  - Highly selective predicate: yields fewer tuples than the hash size.
  - Scans on predicate relation and (most likely) on foreign key relation.
Table Scans During Optimization

Analysis of selection predicates can lead to:

- Full propagation.
  - Highly selective predicate: yields fewer tuples than the hash size.
  - Scans on predicate relation and (most likely) on foreign key relation.
- Updated the bound sketch.
  - Selective predicate but more tuples than hash size.
  - Scan on predicate relation.
Table Scans During Optimization

Analysis of selection predicates can lead to:

- Full propagation.
  - Highly selective predicate: yields fewer tuples than the hash size.
  - Scans on predicate relation and (most likely) on foreign key relation.
- Updated the bound sketch.
  - Selective predicate but more tuples than hash size.
  - Scan on predicate relation.
- Defaulting to unmodified bound sketch.
  - Non selective predicate.
  - Early exit during scan on predicate relation.
Table Scans During Optimization
Table Scans During Optimization

```
movie_companies

<table>
<thead>
<tr>
<th>title</th>
<th>company_name</th>
</tr>
</thead>
<tbody>
<tr>
<td>cast</td>
<td>role</td>
</tr>
<tr>
<td>name</td>
<td>aka</td>
</tr>
</tbody>
</table>
```

Pessimistic Query Optimization
April 23, 2019 39 / 53
1 Query Optimization
2 Motivating Example
3 Prior Work: Cardinality Bounds
4 Tightened Cardinality Bounds
5 Optimizations

6 Evaluation
   - Bound Tightening
   - Runtime Improvement

7 Conclusion and Future Directions
SELECT COUNT(*)
FROM community_44 AS t0,
     community_44 AS t1,
     community_44 AS t2,
     community_44 AS t3
WHERE t0.object = t1.subject AND
     t1.object = t2.subject AND
     t2.object = t3.subject AND
     t0.subject % 512 = 89 AND
     t3.object % 512 = 174;
Googleplus Microbenchmark Examples

```
SELECT COUNT(*)
FROM
  community_30 AS t0,
  community_30 AS t1,
  community_30 AS t2,
  community_30 AS t3,
  community_30 AS t4
WHERE
  t0.object = t1.subject AND
  t0.object = t2.subject AND
  t0.object = t3.subject AND
  t3.object = t4.subject AND
  t0.subject % 256 = 49 AND
  t1.object % 256 = 213 AND
  t2.object % 256 = 152 AND
  t4.object % 256 = 248;
  AND ci.movie_id = mc.movie_id;
```
Googleplus Progressive Bound Tightness

- Postgres
- Bound (Budget 1)
- Bound (Budget 8)
- Bound (Budget 64)
- Bound (Budget 512)
- Bound (Budget 4096)

Relative error: estimate / truth
1. Query Optimization
2. Motivating Example
3. Prior Work: Cardinality Bounds
4. Tightened Cardinality Bounds
5. Optimizations
6. Evaluation
   - Bound Tightening
   - Runtime Improvement
7. Conclusion and Future Directions
Join Order Benchmark

- Built on the IMDb dataset.
  - 113 queries.
  - 33 unique topologies.
  - Skew!
  - Correlation!
  - Complex selection predicates!

Bound Relative Error Versus Postgres Relative Error

- Default Postgres
- Bound (Budget 4096)
Bound Q-Error Versus Postgres Q-Error
Plan Execution Runtime (With Foreign Keys Indexes)

Figure: Linear scale runtime improvements over JOB queries.
Plan Execution Runtime (With Foreign Keys Indexes)

Figure: Linear scale runtime improvements over JOB queries.

- Total runtime.
  - Postgres: 3,190 seconds.
  - Bound (4096 buckets): 1,832 seconds.
Plan Execution Runtime (No Foreign Key Indexes)

Figure: Linear scale plan execution time over JOB queries.
Plan Execution Runtime (No Foreign Key Indexes)

Figure: Linear scale plan execution time over JOB queries.

- Total runtime (including 1 hour cutoff for postgres).
  - Postgres: 21,125 seconds.
  - Bound (4096 buckets): 2,216 seconds.
Takeaways

- Significant gain for very slow queries.
Takeaways

- Significant gain for very slow queries.
- On par with fast queries.
Conclusion and Future Directions

1. Query Optimization

2. Motivating Example

3. Prior Work: Cardinality Bounds

4. Tightened Cardinality Bounds

5. Optimizations

6. Evaluation
   - Bound Tightening
   - Runtime Improvement

7. Conclusion and Future Directions
Optimization Time

- Currently using naive enumeration and sketch construction approach.
Optimization Time

- Currently using naive enumeration and sketch construction approach.
- Approximation of degree statistics.
Contributions

- Technique for tightening theoretically guaranteed join cardinality upper bounds.
Contributions

- Technique for tightening theoretically guaranteed join cardinality upper bounds.
- Method for enumerating practical subset of bounding formulas.
Contributions

- Technique for tightening theoretically guaranteed join cardinality upper bounds.
- Method for enumerating practical subset of bounding formulas.
- Partition budgeting strategy to control the space complexity of our sketches, and the time complexity of our bound calculation.
Contributions

- Technique for tightening theoretically guaranteed join cardinality upper bounds.
- Method for enumerating practical subset of bounding formulas.
- Partition budgeting strategy to control the space complexity of our sketches, and the time complexity of our bound calculation.
- Demonstrate practicality on challenging real world benchmark.
Acknowledgements

- Thank you to Jenny, Tomer, Laurel, Brandon, Jingjing, Tobin, and Leilani!
- This research is supported by NSF grant AITF 1535565 and III 1614738.