HUMAN-POWERED DATA MANAGEMENT

Aditya Parameswaran
with H. Garcia-Molina, J. Widom, A. Polyzotis, M. Teh

Stanford University, MIT, and University of Pennsylvania
Why should we (DM/DB folks) care?

Reason 1: Most data is unstructured

Unstructured Data
- images, videos, text

Structured Data

Automated processing: not yet solved

Incorporate 👥
Why should we (DM/DB folks) care?

Reason 2: S/ware companies use crowds at scale

We undertook a survey of industry crowdsourcing users

Often 10s+ of Millions of $ / yr. / company
(on crowds + supervisors)

Plenty of startups too!
Why should we (DM/DB folks) care?

**Reason 3: Marketplaces are growing rapidly**

- 20+ marketplaces
- Big companies have internal ones

Crowdsourcing Marketplaces

Size of these marketplaces have doubled in 2011 – 2013
Why should we (DM/DB folks) care?

Reason 1: Most data is unstructured

Reason 2: Software companies use crowds at scale

Reason 3: Marketplaces are growing rapidly
What is Human-Powered Data Management?

where humans act as “data processors”
e.g., compare, label, extract

Data Processing Algorithms
Data Processing Systems

Learning accuracies
Interfaces Patterns
Incentives

Machine Learning
HCI
Economics
Efficient Data Processing Algorithms & Systems

Data Processing Algorithms
- Filter [SIGMOD12, VLDB14]
- Clean [KDD12, TKDD13]
- Search [ICDE14]
- Max [SIGMOD12]
- Categorize [VLDB11]
- Debugging [NIPS12]

Data Processing Systems
- Deco [CIKM12, VLDB12, TR12, SIGMOD Record 12]
- DataSift [HCOMP13, SIGMOD14]
- HQuery [CIDR11]

Auxiliary Plugins: Quality, Pricing
- Confidence [KDD13, TR14]
- Pricing [VLDB15]
- Eviction [TR12]
- Quality [HCOMP14]

i.stanford.edu/~adityagp/scoop.html
Data Proc. Sys.: Crowd-Powered Search

Can your search engine handle this?

buildings in the vicinity of

type of cable that connects to

apartments in a good school district near Urbana, with a bus stop near by
DataSift: Crowd-Powered Search

- Non-textual content:
  - “cables that plug into <img>”
  - “funny pictures of cats with hats with captions”

- Time-consuming:
  - “find noise canceling headphones where the battery lasts 13 hrs”
  - “apartments in a nice area around urbana”
<table>
<thead>
<tr>
<th>Rank</th>
<th>Thumbnail</th>
<th>Product Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td><img src="image" alt="Mediabridge Hi-Speed USB 2.0 Cable" /></td>
<td><strong>Mediabridge Hi-Speed USB 2.0 Cable - (6 Feet)</strong>&lt;br&gt;Product page: <a href="http://www.amazon.co/dp/B001MXLD4G">http://www.amazon.co/dp/B001MXLD4G</a>&lt;br&gt;Price: USD 4.99</td>
</tr>
<tr>
<td>4</td>
<td><img src="image" alt="USB Printer Cable for HP DeskJet 1000" /></td>
<td><strong>USB Printer Cable for HP DeskJet 1000 with Life Time Warranty</strong>&lt;br&gt;Product page: <a href="http://www.amazon.co/dp/B004PRXM2C">http://www.amazon.co/dp/B004PRXM2C</a>&lt;br&gt;Reviews: <a href="http://www.amazon.com/reviews/iframe?akid=AKIAJ...">http://www.amazon.com/reviews/iframe?akid=AKIAJ...</a>&lt;br&gt;Price: USD 4.95</td>
</tr>
<tr>
<td>5</td>
<td><img src="image" alt="Mediabridge Hi-Speed USB 2.0 Cable" /></td>
<td><strong>Mediabridge Hi-Speed USB 2.0 Cable - (10 Feet)</strong>&lt;br&gt;Product page: <a href="http://www.amazon.co/dp/B001MSU1HG">http://www.amazon.co/dp/B001MSU1HG</a>&lt;br&gt;Reviews: <a href="http://www.amazon.com/reviews/iframe?akid=AKIAJ...">http://www.amazon.com/reviews/iframe?akid=AKIAJ...</a>&lt;br&gt;Price: USD 5.49</td>
</tr>
<tr>
<td>6</td>
<td><img src="image" alt="Mediabridge Hi-Speed USB 2.0 Cable" /></td>
<td><strong>Mediabridge Hi-Speed USB 2.0 Cable - (16 Feet)</strong>&lt;br&gt;Product page: <a href="http://www.amazon.co/dp/B001MSZBNA">http://www.amazon.co/dp/B001MSZBNA</a>&lt;br&gt;Reviews: <a href="http://www.amazon.com/reviews/iframe?akid=AKIAJ...">http://www.amazon.com/reviews/iframe?akid=AKIAJ...</a>&lt;br&gt;Price: USD 7.49</td>
</tr>
</tbody>
</table>
Building DataSift: Challenges

Ask for text reformulations for query

Check if item satisfies query

- How many reformulations should we gather?
- How many items should we retrieve at each step?
- How do we filter items? How many people do we ask?
- How do we optimize the workflow?
- How do we guarantee correctness?
Fundamental Tradeoffs

- How long can I wait?
- What is my desired quality?
- How much am I willing to spend?

Latency

Cost

Quality
DataSift Summary

Sample applications:

education, social media, commerce, journalism, …

Gather  Retrieve  Filter

Latency

Cost  Quality

[SIGMOD14] DataSift: A Crowd-Powered Search Toolkit (demo)

[HCOMP13] An expressive and accurate crowd powered search
Filtering: The Simplest Version

Dataset of Items → Boolean Predicate → Filtered Dataset

Does X satisfy predicate?

Y Y N

For now, all humans have same error rates

Latency → Quality

Cost
Our Visualization of Strategies

Markov Decision Process

- Yes
- No

- continue
- decide PASS
- decide FAIL
Strategy Examples

- **Yes**
- **No**

- ○ continue
- ● decide PASS
- ◼ decide FAIL
Simplest Version

Given:

- Human error probability (FP/FN)
  - \( \Pr[\text{Yes} | 0]; \Pr[\text{No} | 1] \)
- A-priori probability
  - \( \Pr[0]; \Pr[1] \)

Find strategy with minimum expected cost (\# of questions)

- Expected error < \( t \) (say, 5%)
- Cost per item < \( m \) (say, 20 questions)
Evaluating Strategies

\[ \text{Cost} = \sum (x+y) \Pr[\text{reach}(x,y)] \]

\[ \text{Error} = \sum \Pr[\text{reach} \rightarrow 1] + \sum \Pr[\text{reach} \rightarrow 0] \]

\[ \Pr[\text{reach} (4, 2)] = \Pr[\text{reach} (4, 1) \& \text{get a No}] + \Pr[\text{reach} (3, 2) \& \text{get a Yes}] \]
Naïve Approach

For each grid point
Assign $\bullet$, $\circ$, or $\bullet$

For all strategies:
- Evaluate cost & error
Return the best

$O(3^g)$, $g = O(m^2)$

If $m = 5$, $g = 21$
## Comparison

<table>
<thead>
<tr>
<th>Computing Strategy</th>
<th>Money</th>
</tr>
</thead>
</table>
| Naïve deterministic | Not feasible | $$
| Our best deterministic | Exponential; feasible | $$$

Money costs: $$ for Naïve deterministic strategy, $$$ for Our best deterministic strategy.
Probabilistic Strategy Example

Yes

No

(0.2, 0.8, 0)

continue
decide PASS
decide FAIL
## Comparison

<table>
<thead>
<tr>
<th>Computing Strategy</th>
<th>Money</th>
</tr>
</thead>
<tbody>
<tr>
<td>Naïve deterministic</td>
<td>$$</td>
</tr>
<tr>
<td>Our best deterministic</td>
<td>$$$$</td>
</tr>
<tr>
<td>The best probabilistic</td>
<td>$</td>
</tr>
</tbody>
</table>

**THE BEST**
Finding the Optimal Strategy

Simple: Use Linear Programming

- **variables**: “probabilistic decision per grid point”

- **constraints**:
  - probability conservation
  - boundary conditions

[SIGMOD12] Crowdscreen: Algorithms for filtering data with humans
Generalizations

- Multiple answers (ratings, categories)
- Multiple independent filters
- Difficulty
- Different penalty functions
- Latency
- Different worker abilities
- Different worker probes
- A-priori scores

Doable

Hard!
Generalization: Worker Abilities

<table>
<thead>
<tr>
<th></th>
<th>Item 1</th>
<th>Item 2</th>
<th>Item 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Actual</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>$W_1$</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>$W_2$</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>$W_3$</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

$(W_1 \text{Yes}, \ W_1 \text{No}, …, \ W_n \text{Yes}, \ W_n \text{No})$

$O(m^{2n})$ points

$n \approx 1000$

Explosion of state!
A Different Representation

![Diagram showing a different representation of data with Yes/No and Cost axes, and probabilities denoted by Pr[1|Ans].]
Worker Abilities: Sufficiency

(Yes, No, Yes, No, ..., Yes, No)

Recording Pr[I|Ans] is sufficient:

Strategy  ➔  Optimal
MOOCs: Application of Filtering

Peer Evaluation ≈ Crowdsourcing
Required

Generalization of boolean filtering to scoring [1-5]
Experiments on MOOCs

Stanford HCI Course

1000 x 5 x 5 Parts = 25000 Parts

Graded by random peers with known error rates

To study: how much we can reduce error for fixed cost
Summary:
For same cost, reduction in error (distance from correct grade) of:
- 50% over median
- 30% over MLE
- 10-20% over same accuracy
## Efficient Data Processing Algorithms & Systems

### Data Processing Algorithms
- **Filter** [SIGMOD12, VLDB14]
- **Clean** [KDD12, TKDD13]
- **Search** [ICDE14]
- **Max** [SIGMOD12]
- **Categorize** [VLDB11]
- **Debugging** [NIPS12]

### Data Processing Systems
- **Deco** [CIKM12, VLDB12, TR12, SIGMOD Record 12]
- **DataSift** [HCOMP13, SIGMOD14]
- **HQuery** [CIDR11]

### Auxiliary Plugins: Quality, Pricing
- **Confidence** [KDD13, TR14]
- **Pricing** [VLDB15]
- **Eviction** [TR12]
- **Quality** [HCOMP14]

---

For more information, visit [i.stanford.edu/~adityagp/scoop.html](http://i.stanford.edu/~adityagp/scoop.html)
VISUAL DATA MANAGEMENT with SeeDB

Aditya Parameswaran

with:
Hector Garcia Molina, Sam Madden,
Alkis Polyzotis, Manasi Vartak
Simplifying Data Analytics

Up to a million additional analysts will be needed to address data analytics needs in 2018 in the US alone.

How do we make it easier for novice data analysts to get insights from data?
Data Analytics Workflow

“Production by State”

“Production by Year”

Laborious and Tiresome!
Can we automate this?

Query

Views

“Staplers”

All Products

“Sales by Year”

Similar issues with

Tableau, ShowMe, Profiler, Spotfire
Potentially Interesting Views (Visualizations)

“Potentially interesting”: trend in subset that is not in overall data

Can we automatically highlight potentially interesting views?

Saving: stepping through all views now only potentially interesting ones!
Our Proposed System: SeeDB

DBMS

many rows
many columns

k potentially interesting visualizations

SeeDB

[VLDB14] SeeDB: Visualizing Database Queries Efficiently (Vision)
[VLDB14] Automatically Generating Query Visualizations (Demo)
SeeDB: Conceptual Workflow

Objective: find k-best scoring views (or visualizations)
How do we score views?

This is a hard, domain-specific question!

We are pursuing ways to learn this scoring function using crowds.

For now, a proxy that is “good-enough” differences in “distribution”

e.g., EMD, euclidean, KL-divergence

Difference(Distribution of Sales by year overall, Distribution of Sales by year for Staplers)

our techniques work with any scoring metric
How many views to consider?

Star Schema; Histogram Visualizations

“Sales by Year”

\[ M \text{ measure attributes} \]
\[ A \text{ dimension attributes} \]
\[ F \text{ aggregation measures} \]

One-dimensional visualizations:
\[ M \times A \times F \]

If we consider binning:
\[ M \times A \times F \times B \]
Building SeeDB: Concrete Directions

*How do we minimize computation?*

- Sharing computation
- Approximate visualizations
- Approximate scoring
- Visualization pruning
Technique 1: Sharing Computation

“Sales by Year”  “Production by Year”

“Sales and Production by Year”

SELECT AGG(M1), AGG(M2), D, FROM R
WHERE Prod = “Staplers”
GROUP BY D

Linear Speedup!
Technique 1: Sharing Computation

```
SELECT AGG(M), D1, D2
FROM R
WHERE Prod = "Staplers"
GROUP BY D1, D2
```

Problematic: # of aggregates grow rapidly
Intractable!
Technique 2: Approximate Visualizations

“Production by Year”

Analysts are only interested in trends, not absolutes

Limited also by resolution

Can we provide visualizations that are guaranteed to look similar (e.g., similar order, similar differences) to actual ones, but at much lower cost?
Technique 2: Approximate Visualizations

“Production by Year”

[TR14] Generating Rapid Visualizations with Guarantees

The answer is yes!

At a high-level, algorithm samples “more” from contentious areas

- Order of magnitudes saving compared to baselines
- Optimality guarantees
- Also of independent interest
Building SeeDB: Concrete Directions

How do we minimize computation?

- Sharing computation
- Approximate visualizations
- Approximate utility computation
- Visualization pruning

Overall, a rich space of questions generalizable beyond SeeDB!
Our Current Design

Data scientists rely on visualizations to interpret the data returned by queries, but finding the right visualization remains a manual task that is often laborious. We propose SeeDB, a database visualization engine that partially automates the task of finding the right visualizations for a query. Given an input query Q, SeeDB will efficiently explore not only the space of physical plans for Q, but also the space of possible visualizations for the results of Q. The output will comprise of a recommendation of potentially “insightful” or “interesting” visualizations, where each visualization is coupled with a suitable query execution plan.

II. ARCHITECTURE

SeeDB is implemented as a wrapper around the PostgresSQL database. The architecture of our system is shown in Figure 1 below.

The SeeDB Frontend consists of the Query Builder and the Visualization Engine. It is built on popular JavaScript framework NodeJS (to ensure a responsive user interface and modular code structure), utilizing libraries such as AngularJS (to connect the data model and view), GoogleCharts (to create static graphs), D3 (to create interactive graphs), CrossFilter (to allow subset selection on graphs), and NodeJava (to communicate with the SeeDB Java Backend Engine).

The SeeDB Backend Engine is a highly optimized system that efficiently computes distributions over all combinations of dimension attributes in the queried subset and the entire dataset, as well as the utility scores. The Java powered backend communicates with the frontend by serializing Java objects to JSON.
III. SeeDB FRONTEND

III.1. Query Builder

The SeeDB frontend is designed to allow users with little knowledge of SQL syntax to post queries, retrieve insightful visualizations, and interact with the frontend to drill down on subsets of the result.

The Query Builder (Figure 2) supports two methods of posting queries to the system.

1. Text-based query builder: For more experienced SQL users and users who already know which subset of data is of interest, the frontend provides an interface for entering raw predicates with auto-populated fields to minimize user error (Figure 2, left). It breaks down each predicate into three parts: column name (cand_nm, contbr_st, …), operator (=, >, <, in, …), value (‘CA’, ‘Obama, Barack’, …). The column name dropdown options are automatically populated with the columns designated “dimension” and “measure” when the user selects the table of interest.

Interactive Query Builder!
2. **Graph-based query builder:** For less experienced SQL users and for users who are interested in exploring the entire dataset, the frontend precomputes a distribution graph for the entire dataset over all the columns designated "dimension" (Figure 2, right). The user can simply click-and-drag to select subsets of the graphs. Once a subset is selected, a potential query predicate is generated and displayed below the graphs, the user can then choose to add the predicate to the Query Builder.

The user can also select from a range of distance functions from a dropdown menu. The choice of distance functions is used to compute the utility of each view of queries and rank the produced graphs.

**III.2. Visualization**

Figure 3. SeeDB Frontend Visualization. The visualizations produced are shown to the user in order ranked by utility scores (Figure 3). Each visualization displays the distribution in the query subset (blue) and the distribution in the entire dataset (red). The chart type for each visualization is determined by the dimension attribute.

**Top-k Visualizations**
To summarize...

SeeDB has some ambitious goals...

“show me all that’s interesting about the query result”
i.e., the holy grail of exploratory visual data analysis

We’ve barely scratched the surface, yet!

… doesn’t mean we can’t build a useful tool