INTERACTIVE DATA EXPLORATION

CARSTEN BINNIG
DATA MANAGEMENT LAB
Data Exploration: First step to understand new data sets (overview first and then details)

Vision of Interactive Data Exploration (IDE)

• Intuitive ad-hoc query formulation
• Interactive query execution
• Connect-and-explore for new data sources
DATA EXPLORATION VISION

Avatar (2009)
Siri: Star Trek Computer

DATA EXPLORATION VISION
TODAY’S USER INTERFACES
TODAY’S USER INTERFACES
TODAY’S BIG DATA BACKENDS

Flink
Myria
Spark
Hadoop
Asterix DB
TODAY’S BIG DATA BACKENDS

Low-level Interfaces for ad-hoc Queries

Expensive Data Preparation

Slow & Non-Interactive Execution
VISUAL INTERACTIVE DATA EXPLORATION
What is the closest galaxy?
Challenges & opportunities
CHALLENGE: AD-HOC QUERIES

High-level intuitive ad-hoc query interfaces AND no pre-defined static reports or low-level query interfaces
Response time as small as 500 ms already limit the exploration space and productivity of users

---

The Effects of Interactive Latency on Exploratory Visual Analysis

Zhicheng Liu and Jeffrey Heer

In this research, we have found that interactive latency can play an important role in shaping user behavior and impacts the outcomes of exploratory visual analysis. Delays of 500ms incurred significant costs, decreasing user activity and data set coverage while reducing rates of observation, generalization and hypothesis. Moreover, initial exposure to higher latency interactions resulted in reduced rates of observation and generalization during subsequent analysis sessions in which full system performance was restored.
Users want to directly explore new data **without** heavy-weight data loading, cleaning, indexing, etc.
INCREMENTAL QUERIES

Overview First:

Details on Demand:

Opportunity: Reuse results / compute only the delta
THINK TIME

Response Time after interaction <500ms ... BUT Think Time between interactions > 5-7s

Opportunity: Leverage think time to prepare next step
CONVERSATIONAL INTERFACES

**Traditional:** User asks questions to DBMS...

**Conversational:** DBMS can ask questions back
User: How many people live in Washington?

Systems: Did you mean Washington DC or Washington State?
**VISUAL AND VOICE INPUT/OUTPUT**

**Traditional:** DBMS is agnostic of user + interface

**Opportunity:** Leverage additional knowledge about input/output to optimize execution
OPPORTUNITY: UI CONSTRAINTS

DBMS is aware of UI constraints

Opportunity: Compute only what can be displayed
OPPORTUNITY: HUMAN PERCEPTION

Humans perception is also limited

Opportunity: Compute only what user can perceive
OPPORTUNITY: INTERACTION MODELS

Predicting the next SQL query is hard

User interface limits query interactions

- Linking + brushing
- Zooming
- New attribute
- ...

Opportunity: Better predict what user wants to explore next
OUTLINE

Query Interfaces

Focus: SQL queries and structured data

Other Considerations

- False Discoveries
- Benchmarking
OUTLINE

Query Interfaces

Query Execution

Storage Layer

Other Considerations

• False Discoveries

• Benchmarking
Declarative Query Language for Relational DBMS

SELECT gender, COUNT(*)
FROM customers
WHERE age > 18
GROUP BY gender

Specifies “what” the user wants

DBMS can optimize “how” query is executed
WHY IS SQL NOT ENOUGH FOR IDE?

SQL queries can run forever: no latency guarantees which the user can specify

SQL is agnostic of output rendering (visual / voice): DBMS cannot optimize for the representation to the user

One-shot queries: no incremental query building and conversations

Schema-driven: Users need to know schema to query it (i.e., no ambiguities and error handling, user steering, ...)
QUERY INTERFACES FOR IDE

SQL Extensions for Interactivity

Visual Query Languages

Natural Language Interfaces
  • Incremental Queries
  • Conversational Interfaces
QUERY INTERFACES FOR IDE

SQL Extensions for Interactivity

Visual Query Languages

Natural Language Interfaces
• Incremental Queries
• Conversational Interfaces
Goal: Support interactive SQL aggregate queries

Idea: User specifies latency constraints

SELECT gender, COUNT(*)
FROM customers
WHERE age > 18
GROUP BY gender
WITHIN 500 ms
SPEED/ACCURACY TRADE-OFF

Error

500ms

30 mins

Execution Time (Sample Size)

Interactive Queries

Time to Execute on Entire Dataset
## APPROXIMATE QUERY PROCESSING

### Define Runtime

```sql
SELECT gender, COUNT(*)
FROM customers
WHERE age > 18
GROUP BY gender
WITHIN 500 ms
```

### Define Error

```sql
SELECT gender, COUNT(*)
FROM customers
WHERE age > 18
GROUP BY gender
WITH ERROR 1
WITH CONFIDENCE 95%
```
OTHER APPROXIMATIONS: ORDERING GUARANTEES

Correctness: Ordering guarantees of bars

Rapid Sampling for Visualizations with Ordering Guarantees. (Kim et. al)
QUERY INTERFACES FOR IDE

SQL Extensions for Interactivity

Visual Query Languages

Natural Language Interfaces
  • Incremental Queries
  • Conversational Interfaces
VISUAL QUERY LANGUAGES

Traditional QL:
- User query
  - Query results
  - Visualization

Visual QL:
- Visual specifications
  - Transformations to pixel space visual optimizations
  - Reduced rendering time
  - Logical visual plans \(\rightarrow\) Physical query plans

*Overview of Data Exploration Techniques, Tutorial (Idreos et al.)*
query := (data, mark, encoding)

(a) Line chart with aggregation

```json
{   "data" = SELECT
    temp_max, date,
    location FROM weather }
"mark": "line",
"encoding": {
    "x": {
        "field": "date",
        "type": "temporal",
        "timeUnit": "month" },
    "y": {
        "field": "temp_max",
        "type": "quantitative",
        "aggregate": "mean" },
    "color": {
        "field": "location",
        "type": "nominal" }
}
```
EXAMPLE: VEGA-LITE

query := (data, mark, encoding, vis-params)

(a) Line chart with aggregation

```json
{   "data" = SELECT
    temp_max, date,
    location FROM weather }
"mark": "line",
"encoding": {   "x": {   "field": "date",
"type": "temporal",
"timeUnit": "month" },
"y": {   "field": "temp_max",
"type": "quantitative",
"aggregate": "mean" },
"color": {   "field": "location",
"type": "nominal" }
"width": 200,
"height": 200
}
```

Vega-Lite: A Grammar of Interactive Graphics (Satyanarayan et. al)
APPORXIMATION: PIXEL CORRECTNESS

RDBMS

data reduction
data

data flow

Query Rewriter

reduction
query

query
visualization
parameters

Visualization
Client

data-reduced query result

selected time range

a) baseline vis. of Q

foreground pixels

background pixels

b) vis. of MinMax(Q)

inter-group line

c) vis. of M4(Q)

inner-group lines

M4: A Visualization-Oriented Time Series Data Aggregation (Jugel et. al.)
APPROXIMATION: PERCEPTION
SELECT gender, COUNT(*)
FROM customers
WHERE age > 18
GROUP BY gender

RENDERED BY HISTOGRAM
SIZE 200,200
PERCEIVED BY JND

Towards Perception-aware Interactive Data Visualization Systems (Eugene Wu et. al)
QUERY INTERFACES FOR IDE

SQL Extensions for Interactivity

Visual Query Languages

Natural Language Interfaces
  • Incremental Queries
  • Conversational Interfaces
User: How many people live in Munich?

Systems: There are 1.43 Million people living in Munich

User: And how many of them are female?
NLDB: INCREMENTAL QUERIES

Stateless NLQs

- NLQ
- NLQ Engine
- Databases

Each query must be
- Fully specified
- Processed independently

Stateful NLQs

- NLQ
- NLQ Engine
- Databases
- Query history

Each query
- Can be partially specified
- Processed with regards to previous queries

Natural Language Data Management and Interfaces: Recent Development and Open Challenges, SIGMOD 2015 Tutorial (Li et. al)
**NLDB: CONVERSATIONAL INTERFACE**

![Diagram of NLDB conversational interface](image)

**Feedback Generation**

**Query Understanding**

**Domain knowledge**

**Query Translation**

**Data store**

Semantic Parse Tree

SQL, SPARQL, ...

---

*Natural Language Data Management and Interfaces: Recent Development and Open Challenges, SIGMOD 2015 Tutorial (Li et. al)*
NLIDB: AD-HOC QUERIES

Low

- Query naturalness
- Grammar complexity
- Vocabulary complexity
- Ambiguity
- Parser error

High

Ad-hoc NLQs

Natural Language Data Management and Interfaces: Recent Development and Open Challenges, SIGMOD 2015 Tutorial (Li et. al)
NLIDB: AMBIGUITIES AND ERRORS

Schema ambiguities (table and attribute names)

“Doctor” vs. “Physician”

Data ambiguities

“NYC” vs. “New York” vs. “New York City”

Missing / imprecise information

“How many old people are there?”

Paraphrasing (syntactical, lexical, …)

“Show me …” vs. “What are …” vs. “I want to see”
NLIDB: ERROR HANDLING

No Error Handling

Interactive (Conversational):
- Disambiguate parts of queries (e.g., table / attribute names)
- List query alternatives (e.g., explain SQL queries to user)

Automatic:
- Machine learning (e.g., named entity recognition, paraphrase detection, …)
- Additional domain knowledge (e.g., ontologies, wordnet, …)
OUTLINE

Query Interfaces

Query Execution

Storage Layer

Other Considerations
• False Discoveries
• Benchmarking
SQL QUERY EXECUTION

SQL Query

Exact Answer
SQL EXECUTION TIME

Run time

Data Size

Execution Time

Interactivity Threshold
**SQL QUERY EXECUTION**

```
SELECT c.name, SUM(o.total)
FROM customer c JOIN orders o
WHERE c.age > 18
GROUP BY c.name
```

SQL aggregate queries (SUM, COUNT, AVG, ...) are blocking!
PROBLEMS WITH SQL EXECUTION

Not interactive: Execution time of SQL aggregate queries increases with data size

Not incremental: Queries are executed independently / no direct support for sessions

No optimization for user and interfaces: Optimization only takes query and data as input

No conversations and steering: only query + data in and result out
QUERY EXECUTION FOR IDE

Interactive Query Processing

Support for Incremental Queries

Predicting User Behavior
QUERY EXECUTION FOR IDE

Interactive Query Processing

Support for Incremental Queries

Predicting User Behavior
TRICKS WE ALL KNOW BUT ...

1PB on 100 machines

hours minutes 500ms?

Hard Disks Memory ?
APPROXIMATE QUERY PROCESSING (AQP)
WHY APPROXIMATION IS USEFUL

Large data warehouses

- Gigabytes to terabytes of data

Query characteristics:

- Access large fraction of database
- Queries: aggregate queries
- Absolute precision unnecessary
  - $89,000 after 5 secs vs. $89,034.57 after 2 hrs
**BASIC IDEA OF AQP**

<table>
<thead>
<tr>
<th>Product</th>
<th>Amount</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU</td>
<td>1</td>
</tr>
<tr>
<td>CPU</td>
<td>1</td>
</tr>
<tr>
<td>CPU</td>
<td>2</td>
</tr>
<tr>
<td>CPU</td>
<td>3</td>
</tr>
<tr>
<td>CPU</td>
<td>4</td>
</tr>
<tr>
<td>Disk</td>
<td>1</td>
</tr>
<tr>
<td>Disk</td>
<td>2</td>
</tr>
<tr>
<td>Monitor</td>
<td>1</td>
</tr>
</tbody>
</table>

**Sales**

<table>
<thead>
<tr>
<th>Product</th>
<th>Amount</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU</td>
<td>1</td>
</tr>
<tr>
<td>CPU</td>
<td>2</td>
</tr>
<tr>
<td>CPU</td>
<td>3</td>
</tr>
<tr>
<td>Disk</td>
<td>2</td>
</tr>
</tbody>
</table>

**SalesSample**

<table>
<thead>
<tr>
<th>Product</th>
<th>Amount</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU</td>
<td>1</td>
</tr>
<tr>
<td>CPU</td>
<td>2</td>
</tr>
<tr>
<td>CPU</td>
<td>3</td>
</tr>
<tr>
<td>Disk</td>
<td>2</td>
</tr>
</tbody>
</table>

**SELECT SUM(Amount) FROM Sales WHERE Product = 'CPU'**

- **Exact Answer:**
  \[1+1+2+3+4 = 11\]

- **Approx. Answer:**
  \[(1+2+3)\times 2 = 12\]
APPROACH 1: ONLINE SAMPLING

Main Ideas:

• Sampling at query time (online sampling)
• Query answers continually improve
• Returns exact result at the end
APPROACH 2: OFFLINE SAMPLING

Main Ideas:

• Construct & store synopses prior to query time
• At query time, use synopses to answer the query

Differences to Online Aggregation:

• Better I/O behavior BUT
• Need to maintain synopses up-to-date
• Need to know query workload in some cases (e.g., for stratified samples)
ONLINE AGGREGATION

Don’t process in batch!

Use a pipelined query execution model

Compute aggregation online while streaming over data:

*Online Aggregation (Hellerstein et. al.)*
ISN‘T THIS JUST SAME AS SAMPLING?

Yes (it is online sampling)

• We need values to arrive in random order (same as offline samples)!
• “Confidence intervals” similar to offline-sampling work

... and No!

• stopping condition can be set on the fly
• can handle queries w/o prior knowledge
ONLINE AGGREGATION WITH GROUPING

Query: SELECT online_avg(grade) FROM ENROLL GROUP BY major;

![Postgres95 Online Aggregation Interface]

<table>
<thead>
<tr>
<th>major</th>
<th>AVG</th>
<th>Confidence</th>
<th>Interval</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2.27216</td>
<td>95</td>
<td>0.160417</td>
</tr>
<tr>
<td>2</td>
<td>2.56146</td>
<td>95</td>
<td>0.160417</td>
</tr>
<tr>
<td>3</td>
<td>2.66702</td>
<td>95</td>
<td>0.160417</td>
</tr>
<tr>
<td>4</td>
<td>2.86235</td>
<td>95</td>
<td>0.160417</td>
</tr>
<tr>
<td>5</td>
<td>3.12048</td>
<td>95</td>
<td>0.160417</td>
</tr>
<tr>
<td>9</td>
<td>2.89645</td>
<td>95</td>
<td>0.160417</td>
</tr>
</tbody>
</table>
FAIRNESS/MISSING GROUPS

Make sure that all groups progress at the same speed

Open issue: missing groups (outliers)
A NAÏVE APPROACH

Query: SELECT online_avg(grade) FROM ENROLL;

Can do it in most DBMSs today!

Naïve Idea: Implement as user-defined-function

• But …can’t meet basic requirements:
  • No continuous output
  • No random access to data
  • No fairness (or control over partiality)
RANDOM ACCESS TO DATA

Table Scan

- OK if clustering attribute is uncorrelated to aggregate or grouping attributes

Index Scan

- OK if index attribute(s) are uncorrelated to aggregate or grouping attributes
- Or index on random()? 

Online Sampling:

- Could introduce new sampling access methods to databases (e.g. Frank Olken’s work)
FAIRNESS WITH INDEX SCANS

Index Striding: *random tuple from Group 1, random tuple from Group 2, ...*

- Access method opens many cursors in index, one per group.
- Fetch round-robin.
- Can control speed by weighting
- Gives fairness/partiality, info/speed match!

Problem for IDE: Can not index all attributes and combinations (i.e., for multiple group-by atts)
ESTIMATE THE AGGREGATE

Query: SELECT online_agg(expr) FROM Table

AVG(expr):

\[
\overline{Y}_n = \frac{1}{n} \sum_{i=1}^{n} v(L_i)
\]

- \(v(L_i)\): value of expr for \(i\)-th random tuple
- \(n\) tuples seen so far of \(m\) tuples in table

Can be extended to other aggregates (SUM, COUNT, STDEV, …) and queries with a WHERE clause
CONFIDENCE INTERVALS

Confidence Intervals for aggregate queries

- given an estimate + a probability $p$
- Return interval that we’re within error $e$ of the right answer

Different types of estimates

① Conservative (Hoeffding): $n \geq 1$
② Large-Sample (CLT): $n$ small and large enough!
   Smaller intervals than conservative
CONSERVATIVE INTERVALS

Query: SELECT online_agg(expr) FROM Table

Conservative Intervals:
• a, b: min and max of aggregate expression
• p: probability of value being in interval
• n: number of tuples seen so far

AVG(expr):
\[ \epsilon_n = (b - a) \left( \frac{1}{2n} \ln \left( \frac{2}{1 - p} \right) \right)^{1/2} \]
JOINS AND ONLINE AGGREGATION

Sort-merge and hash join algorithms are blocking algorithms – not acceptable in online aggregation

Merge (without sorting) provides ordered output – bad for statistical estimator in online aggregation

Nested loop join is the best, but...
AN ARTIFICIAL EXAMPLE

Query: SELECT AVG(S.a + R.b) FROM R, S
GROUP BY R.c

If R is the inner relation, for each random tuple from S, we need to scan the whole relation R

• Interactive threshold might not be met
• Can not guarantee randomness, fairness on R
OVERVIEW OF RIPPLE JOIN

Select tuples from R and S interchangeable at random

Ripple Joins for Online Aggregation (Haas et. al.)
RIPPLE JOIN ALGORITHMS

It can be viewed as a generalization of nested loops join

Roles of "inner" and "outer" relation are continually interchanged during processing
RIPPLE JOIN VARIANTS

Block ripple join improves performance on I/O

Indexed ripple join

• index-enhanced nested-loops join
• the role of inner/outer relations does not alternate any more

Hash ripple join can be used for equijoin queries – two hash tables in memory for both R and S
AQP WITH PRE-COMPUTED SAMPLES

Step 1: Sample creation (offline)
• Build a set of multi-dimensional samples
• Create different sizes to meet different constraints

Step 2: Sample selection (at runtime)
• Selects best sample based on query’s accuracy or response time requirements
• Use an Error-Latency-Profile heuristic
SAMPLING CREATION: UNIFORM VS. STRATIFIED

Uniform sampling / random samples

• Sample random tuples from database (e.g., reservoir sampling)

• Groups with few entries would have significantly lower confidence bounds than popular data

• Problem for IDE: Might miss rare groups entirely

Stratified sampling:

• Rare subgroups are over-represented relative to a uniform sample

• Problem for IDE: Needs to know workload
SAMPLE SELECTION

Goals:
- Meet runtime
- Minimize error

SELECT gender, COUNT(*)
FROM customers
WHERE age > 18
GROUP BY gender
WITHIN 5 seconds
QUERY EXECUTION FOR IDE

Interactive Query Processing

Support for Incremental Queries

Predicting User Behavior
INCREMENTAL QUERY INTERFACES

SELECT Category, SUM(Sales) FROM T WHERE 300 < Sales < 700 GROUP BY Category

SELECT Category, SUM(Sales) FROM T WHERE 200 < Sales < 600 GROUP BY Category

SELECT Category, SubCat, SUM(Sales) FROM T WHERE 200 < Sales < 600 GROUP BY Category, SubCat
1. **Reason about reuse** potential of intermediate results

2. **Add additional materialization** operations to query plan

3. Analyze if subsequent query can **reuse cached intermediates**

---

**Unclear if extra costs pay off in future**
Main idea: Reuse internal structures
- Keep internal data structures
- Reuse for subsequent queries

Savings are two-fold
- No additional materialization costs
- No need to re-create internal structures

More robust towards different reuse-potentials
Workloads: 64 queries with different reuse potentials

- **Low**: 1% of the cached data is reused on average
- **Medium**: 10% of the cached data is reused on average
- **High**: 50% of the cached data is reused on average
QUERY EXECUTION FOR IDE

Approximate Query Processing

Support for Incremental Queries

Predicting User Behavior
PREDICTING USER BEHAVIOR

DBMS

Query Formulation → Wait time → Result Review

Wait time → Query Execution → Idle time

Idle time → Query Execution → Speculation

Speculative Execution

Ideal: No wait time

Follow-up Query → Result Review

Query Formulation → Result Review

Query Execution → Speculation

Potential Follow-up Query
SPECULATIVE EXECUTION

exploration space reduction → query enumeration → query ranking

Distributed and interactive cube exploration (Karnat et. al.)
**SPECULATIVE EXECUTION**

**User query**

```
SELECT AVG(iops) FROM events
WHERE month="m1" AND week="w1"
GROUP BY zone
```

**Cube exploration operators**

- `WHERE month="m1"`
- `WHERE month="m1" AND week="w1" AND hour="h1"`
- `WHERE month="m1" AND week="w2"`

**Exploration space reduction**
SPECULATIVE EXECUTION

exploration space reduction → query enumeration

**user query**

```
SELECT AVG (iops) FROM events
WHERE month="m1" AND week="w1"
GROUP BY zone
```

**speculative queries**

```
Q(month="m1")
...
Q(month = "m12")
...
Q(hour ="h1")
...
Q(hour ="h24")
...
Q(week="w2")
...
Q(week="w3")
```
SPECULATIVE EXECUTION

**user query**

```
SELECT AVG (iops) FROM events
WHERE month="m1" AND week="w1"
GROUP BY zone
```

**speculative queries**

```
Q(month="m1")
...
Q(month = "m12")
...
Q(hour ="h1")
...
Q(hour =" h24")
Q(week="w2")
...
Q(week="w3")
```
OUTLINE

Query Interfaces

Query Execution

Storage Layer

Other Considerations
  • False Discoveries
  • Benchmarking
STORAGE SUPPORT FOR ...

Approximate Computing / Online Sampling

Incremental query processing

Visual results
**VISTREE: INDEX FOR VISUAL DATA EXPLORATION**

**Main Design Goals:**

- Fast computation of **binned plots** (histograms, pie charts, ...)
- Native support of **user interactions**
- Interactive response time < **500ms**
IMPORTANT USER INTERACTIONS

“Overview first, zoom and filter, then details-on-demand.” [Shneiderman, 1996]

Overview:

Default Binning

Brushing-and Linking:

Filtering on bins

Zooming:

Refine Binning
IDEA 1: VISUALLY-BALANCING

Annotations (e.g., COUNTs)

1st vis-level (l=0)

2nd vis-level (l=1)

Other levels (l>1)

Buckets (Vis.-aligned)

Visually-balanced

Normally-balanced
EXAMPLE: VISUALLY-BALANCING

Histograms:

VisTree: 1st vis-level

Visually-aligned

Salary Histogram
IDEA 2: HISTOGRAM LOOKUPS

Batch lookups $H=<\text{Buckets, Filters}>$

1st vis-level ($l=0$)

2nd vis-level ($l=1$)

Other levels ($l>1$)

- Visually-balanced
- Normally-balanced

Buckets (Vis.-aligned)
Linking Brushing:

VisTree: 1st vis-level

Salary Buckets
IDEA 2: HISTOGRAM LOOKUPS

Batch lookups $H = \langle \text{Buckets, Filters} \rangle$

1st vis-level ($l=0$)

2nd vis-level ($l=1$)

Stateful Search

Other levels ($l>1$)

Perceptual-Aware Approximation

- Visually-balanced
- Normally-balanced
Perceptual-Aware Approximation:

- Progressive & approximate **top-down traversal**
- **Stop if visual change is not perceived by user** (e.g., we use Just-Noticeable-Difference JND as perceptual function)
IDEA 3: INCREMENTAL INDEXING

Building all possible VisTrees is too expensive

Incremental Indexing: 1-dim VisTrees -> 2-dim -> 3-dim -> ...

1. Create. VisTrees on-demand

2. Prioritize required parts

3. Histogram lookup during creation
OUTLINE

Query Interfaces

Query Execution

Storage Layer

Other Considerations
  • False Discoveries
  • Benchmarking
MANY OTHER TOPICS

Data Integration and Cleaning

Other Workloads (ML, Graphs, Streaming, Text, …)

Risk of False Discoveries

Benchmarking

Hardware Support (Approximate HW, Fast Networks, Specialized Co-Processors, …)

…
OUTLINE

Query Interfaces

Query Execution

Storage Layer

Other Considerations

- False Discoveries
- Benchmarking
A New Study shows: A Glass Of Red Wine Is The Equivalent To An Hour At The Gym

http://www.huffingtonpost.co.uk/2016/01/08/a-glass-of-red-wine-is-the-equivalent-to-an-hour-at-the-gym-says-new-study_n_7317240.html
is associated with a reduced risk of lung cancer
"A false discovery rate of over 80%"

"Any claim coming from an observational study is most likely to be wrong."

...
ARE WE CONTRIBUTING TO THE PROBLEM?
CONTROLING FALSE DISCOVERIES

Towards Sustainable Insights
or why polygamy is bad for you
Carsten Binnig Lorenzo De Stefani Tim Kraska Eli Upfal
Department of Computer Science, Brown University
firstname.lastname@brown.edu

ABSTRACT
Have you ever conducted a test that produced non-significant results and thought it a pity, especially if the test was conducted for a widely publicized grant proposal? We refer to this phenomenon as false discovery. We propose several novel techniques to automatically suggest visualizations, correlations, and perform visual data exploration, significantly increase the chance that a user makes a false discovery like this one. In this paper, we first show how current tools mislead users to consider random fluctuations as significant discoveries. We then describe our vision and early results for Quide, a new system for automatically controlling the various risk factors during the data exploration process.

Controlling False Discoveries During Interactive Data Exploration
Zheguang Zhao Lorenzo De Stefani Emanuele Zgraggen Carsten Binnig
Department of Computer Science, Brown University
firstname.lastname@brown.edu

ABSTRACT
Recent tools for interactive data exploration automatically provide the user with new visualizations and suggest potential correlations. However, these tools often fail to account for the high risk of false discovery. We demonstrate the importance of controlling the false discovery rate (FDR) and present a new system called Quide, which automatically controls the FDR during the data exploration process. Quide provides users with a visual summary of the data and suggests potential correlations and visualizations that are statistically significant. We conducted a user study with 20 participants to evaluate the effectiveness of Quide and found that users were significantly more likely to discover true correlations when using Quide compared to the standard system. Our results show that Quide significantly reduces the risk of false discovery and improves the overall quality of the data exploration process.

Vision Paper
Avoiding Statistical Pitfalls

What you see is not what you get!
Detecting Simpson’s Paradoxes during Data Exploration
Yue Guo, Carsten Binnig, Tim Kraska
Brown University Providence, USA
TU-Darmstadt Darmstadt, Germany

ABSTRACT
Visual data exploration tools, such as Vizor or Tablore, significantly simplify data exploration for domain experts and, more importantly, reduce complex and counterintuitive differences between various studies by means of an interactive and dynamic visualization. However, there are many statistical pitfalls in data analysis. As a result of this expertise [9], there exists a paradox that leads to the wrong conclusion when splitting the visualized data into subgroups. In this work, we present a new tool that automatically detects and highlights Simpson’s Paradox, which is a special type of error in which a high-level aggregate visualization leads to the wrong conclusion. Our tool continuously monitors and updates the statistical analysis even when the user explores different subgroups of the data. The tool highlights the statistical significance of differences between the subgroups and provides users with clear and concise information about the magnitude of the effect.

Avoiding Statistical Pitfalls

SIGMOD 2017
CIDR 2017
HILDA 2017
OUTLINE

Query Interfaces

Query Execution

Storage Layer

Other Considerations

• False Discoveries
• Benchmarking
BENCHMARKING INTERACTIVE DATA EXPLORATION (BIDE)

DB Community

- use analytical benchmarks (e.g., TPC-H)
- use non-incremental queries
- not user-focused
- report the performance of single queries

Viz Community

- studies are user-focused
- use insight-based evaluation methods
- costly, difficult to compare/reproduce
- introduces UI bias

BIDE

- evaluates different aspects of IDE workflows
- simulates common user behavior
- has no UI bias
- easy to compare/reproduce
- Works for batch and approximate systems
COMMON DATA EXPLORATION TASKS

- Visual Analysis
- Model Building
- Recommendations
- Data Cleaning
- Statistical Testing
FIRST STEP – BIDE 1.0

- BIDE 1.0 is a benchmark for Visual Analysis workflows in IDE systems
- Provides a simulator to mimic common user behavior
- Focus on incremental query building
Request histogram $V_1$ of all flight departure delays
[wait for $n$ seconds]
Request histogram $V_2$ of all flight arrival delays
[wait for $n$ seconds]
Filter $V_1$ and $V_2$ by airline AA
[wait for $n$ seconds]
Filter $V_1$ and $V_2$ by airline DL
[wait for $n$ seconds]
Request a scatterplot $V_3$ of all departure and arrival delays
[wait for $n$ seconds]
Filter $V_1$, $V_2$, $V_3$ by all direct flights between RI and WA
## BIDE Result Metrics

<table>
<thead>
<tr>
<th></th>
<th>MONETDB</th>
<th>IDEA</th>
<th>XDB</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Undiscovered Bins</td>
<td>Unprocessed Tuples</td>
<td>Total Error</td>
</tr>
<tr>
<td><strong>S</strong></td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td></td>
<td>+/-0.00</td>
<td>+/-0.00</td>
<td>+/-0.00</td>
</tr>
<tr>
<td><strong>M</strong></td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td></td>
<td>+/-0.00</td>
<td>+/-0.00</td>
<td>+/-0.00</td>
</tr>
<tr>
<td><strong>S</strong></td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td></td>
<td>+/-0.00</td>
<td>+/-0.00</td>
<td>+/-0.00</td>
</tr>
<tr>
<td><strong>M</strong></td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td></td>
<td>+/-0.00</td>
<td>+/-0.00</td>
<td>+/-0.00</td>
</tr>
<tr>
<td><strong>S</strong></td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td></td>
<td>+/-0.00</td>
<td>+/-0.00</td>
<td>+/-0.00</td>
</tr>
<tr>
<td><strong>M</strong></td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td></td>
<td>+/-0.00</td>
<td>+/-0.00</td>
<td>+/-0.00</td>
</tr>
</tbody>
</table>

[Latency Requirement: 500ms, Data Size: medium, Schema: denormalized]
SUMMARY

Database Architecture has to change for IDE

• Query Interfaces
• Query Execution
• Storage Layer

Many other research topics

• False Discoveries
• Benchmarking
• …