# Fast Algorithms for Mining Association Rules

Rakesh Agrawal      Ramakrishnan Srikant
IBM Almaden Research Center
VLDB 1994

Slides adopted from Dan Li, Sudeepa Roy

Presenter: Yingfeng Lu

Discussion Leader: Jianhao Cao

CPSC 504

# Data Mining Definition

Data mining is the exploration and analysis of large quantities of data in order to discover valid, novel, potentially useful, and ultimately understandable patterns in data.

- Valid: The patterns hold in general.
- Novel: We did not know the pattern beforehand.
- Useful: We can devise actions from the patterns (business intelligence).
- Understandable: We can interpret and comprehend the patterns

# Characteristics of Data Mining

- Key Characteristics
  - Large, multidimensional datasets
  - Efficient algorithms to “discover” knowledge
- What’s the connection with database systems?
  - Managing the data
    - Extract, Transform, and Load
    - There may be many distributed, heterogeneous sources.
  - Large numbers of records;  many dimensions
  - Heavy emphasis on I/Os
  - Query evaluation and optimization considerations

# Association Rules


- One type of data mining rules is Association Rules
- An example rule is “people who buy diapers tend to buy beer”
- This is useful for stores because they can improve stock
- They’ve also been used in many areas, including medical diagnoses, protein sequence composition, health insurance claim analysis and census data

# Here’s the plan

- Stores keep track of all the items that people bought at a time
- By looking at all of the different purchases, we can figure out which items were bought at the same time
- Then we can figure out which one was the “cause” and which one was the “effect”

# Notations & Sample Data

- Notations:
  - Items: $I = \{i_1, i_2, \dots, i_m\}$
  - $D$: a set of transactions
  - A transaction: $T$, $T \subseteq I$ , with a unique identifier TID

- Each row is a *transaction* – one person's grocery order
- So in T2 the person bought Sushi and Bread

| | |
|---|---|
| T1 | Sushi, Chicken, Milk |
| T2 | Sushi, Bread |
| T3 | Bread, Vegetables |
| T4 | Sushi, Chicken, Bread |
| T5 | Sushi, Chicken, Ramen, Bread, Milk |
| T6 | Chicken, Ramen, Milk |
| T7 | Chicken, Milk, Ramen |

- Now we need to decide whether there are any items that people tend to buy when they buy other items. We refer to this as a rule

# Support

- Informally: support measures if items appear together a lot of times
- Formally: A rule X→Y holds with support *s* if s% of transactions contain *X* **AND** *Y*.

| | |
|---|---|
| T1 | Sushi, Chicken, Milk |
| T2 | Sushi, Bread |
| T3 | Bread, Vegetables |
| T4 | Sushi, Chicken, Bread |
| T5 | Sushi, Chicken, Ramen, Bread, Milk |
| T6 | Chicken, Ramen, Milk |
| T7 | Chicken, Milk, Ramen |

- For example, {Chicken, Ramen, Milk} occurs with 3/7= 42% support

# Confidence

- Informally: confidence measures which items suggest the others will be there, too.
- Formally: A rule X→Y holds with *confidence* c if c% of transactions that contain X also contain Y

| | |
|---|---|
| T1 | Sushi, Chicken, Milk |
| T2 | Sushi, Bread |
| T3 | Bread, Vegetables |
| T4 | Sushi, Chicken, Bread |
| T5 | Sushi, Chicken, Ramen, Bread, Milk |
| T6 | Chicken, Ramen, Milk |
| T7 | Chicken, Milk, Ramen |

Ramen → Milk, Chicken [conf = 3/3 = 100%]

Ramen, Chicken → Milk [conf = 3/3 = 100%]

# When is a rule valid?

A rule is valid if its support is above a given threshold (minimum support) and its confidence is over another given threshold (minimum confidence).

A frequent itemset (or large itemset) is a set of items that has at least minimum support

| | |
|---|---|
| T1 | Sushi, Chicken, Milk |
| T2 | Sushi, Bread |
| T3 | Bread, Vegetables |
| T4 | Sushi, Chicken, Bread |
| T5 | Sushi, Chicken, Ramen, Bread, Milk |
| T6 | Chicken, Ramen, Milk |
| T7 | Chicken, Milk, Ramen |

In this example, {chicken, milk, ramen} is a frequent itemset if the minimum support is less than 3/7.

# Problem Decomposition

The problem of discovering all association rules can be decomposed into two subproblems:

- 1. Find all sets of items (itemsets) that have transaction support above minimum support.
  - Two algorithms: Apriori and AprioriTid
- 2. Use the large itemsets to generate the desired rules.

The paper focuses on subproblem 1.

# Key idea of Apriori

Calculating association rules on terabytes of data can be sloooowww. The slowest part is *counting the support*.

The Apriori algorithm speeds things up based on the observation that each subset of a frequent itemset must *also* be a frequent itemset

For example, since rice only appears one time, it can't appear 2 or times with anything else.

| Transaction | Items |
|---|---|
| T1 | apple, dates, rice, corn |
| T2 | corn, dates, tuna |
| T3 | apple, corn, dates, tuna |
| T4 | corn, tuna |

# Discovering Large Itemsets

Intuition: any subset of a large itemset must be large.

Algorithms for discovering large itemsets make multiple passes over the data.

- In the first pass, determine which individual item is large.
- In each sequent pass,
  - Previous large itemsets are used to generate candidate itemsets.
  - Count actual support for the candidate itemsets.
  - Determine which are the real large itemsets.
- This process continues until no new large itemsets are found.

# Algorithm Apriori

$L_1 = \{\text{large 1-itemsets}\};$
**for** ( $k = 2;\ L_{k-1} \neq \emptyset;\ k{+}{+}$ ) **do begin**
    $C_k = \text{apriori-gen}(L_{k-1});$ // New candidates
    **forall** transactions $t \in \mathcal{D}$ **do begin**
        $C_t = \text{subset}(C_k, t);$ // Candidates contained in $t$
        **forall** candidates $c \in C_t$ **do**
            $c.\text{count}{+}{+};$
    **end**
    $L_k = \{c \in C_k \mid c.\text{count} \geq \text{minsup}\}$
**end**
Answer $= \bigcup_k L_k;$

# apriori-gen function

Input: large itemsets $L_{k-1}$
Output: New candidate itemsets $C_k$

- The apriori-gen function has two steps:
  - Join
  - Prune

# Example for Apriori

Round 1: Find all large itemsets of size 1

Start by finding the support of all itemsets of size 1

Support: {apple} = 2/4
{corn} = 4/4
{dates} = 3/4
{rice} = 1/4
{tuna} = 3/4

| Transaction | Items |
|---|---|
| T1 | apple, dates, rice, corn |
| T2 | corn, dates, tuna |
| T3 | apple, corn, dates, tuna |
| T4 | corn, tuna |

Minimum support = 50%

# Example for Apriori

## Round 2: Find all large itemsets of size 2

All possible itemsets of size 2:

{apple, corn}
{apple, dates}
{apple, rice}
{apple, tuna}
{corn, dates}
{corn, rice}
{corn, tuna}
{dates, rice}
{dates, tuna}
{rice, tuna}

| Transaction | Items |
|---|---|
| T1 | apple, dates, rice, corn |
| T2 | corn, dates, tuna |
| T3 | apple, corn, dates, tuna |
| T4 | corn, tuna |

Minimum support = 50%

Because {rice} only occurs once, anything including {rice} can't occur 2 or more times, so we can ignore itemsets including {rice}.

# Example for Apriori

## Round 2: Find all large itemsets of size 2

All possible itemsets of size 2:

{apple, corn}
{apple, dates}
~~{apple, rice}~~
{apple, tuna}
{corn, dates}
~~{corn, rice}~~
{corn, tuna}
~~{dates, rice}~~
{dates, tuna}
~~{rice, tuna}~~

| Transaction | Items |
|---|---|
| T1 | apple, dates, rice, corn |
| T2 | corn, dates, tuna |
| T3 | apple, corn, dates, tuna |
| T4 | corn, tuna |

Minimum support = 50%

Because {rice} is not frequent, anything including {rice} is not frequent, so we can ignore itemsets including {rice}.

# Example for Apriori

## Round 2: Find all large itemsets of size 2

All possible itemsets of size 2:

{apple, corn}
{apple, dates}
~~{apple, rice}~~
{apple, tuna}
{corn, dates}
~~{corn, rice}~~
{corn, tuna}
~~{dates, rice}~~
{dates, tuna}
~~{rice, tuna}~~

| Transaction | Items |
|---|---|
| T1 | apple, dates, rice, corn |
| T2 | corn, dates, tuna |
| T3 | apple, corn, dates, tuna |
| T4 | corn, tuna |

Minimum support = 50%

# Example for Apriori

## Round 2: Find all large itemsets of size 2

Support for all possible itemsets of size 2:

{apple, corn} = 2/4
{apple, dates} = 2/4
~~{apple, rice}~~
{apple, tuna} = 1/4
{corn, dates} = 3/4
~~{corn, rice}~~
{corn, tuna} = 3/4
~~{dates, rice}~~
{dates, tuna} = 2/4
~~{rice, tuna}~~

| Transaction | Items |
|---|---|
| T1 | apple, dates, rice, corn |
| T2 | corn, dates, tuna |
| T3 | apple, corn, dates, tuna |
| T4 | corn, tuna |

Minimum support = 50%

# Example for Apriori

## Round 2: Find all large itemsets of size 2

All frequent itemsets of size 2:

{apple, corn}
{apple, dates}
{corn, dates}
{corn, tuna}
{dates, tuna}

| Transaction | Items |
|---|---|
| T1 | apple, dates, rice, corn |
| T2 | corn, dates, tuna |
| T3 | apple, corn, dates, tuna |
| T4 | corn, tuna |

Minimum support = 50%

# Example for Apriori

## Round 3: Find all large itemsets of size 3

Given frequent itemsets of size 2

{apple, corn}
{apple, dates}
{corn, dates}
{corn, tuna}
{dates, tuna}

| Transaction | Items |
|---|---|
| T1 | apple, dates, rice, corn |
| T2 | corn, dates, tuna |
| T3 | apple, corn, dates, tuna |
| T4 | corn, tuna |

Minimum support = 50%

Find out all the possible frequent itemsets of size 3:

{apple, corn, dates}
{corn, dates, tuna}

# Example for Apriori

## Round 3: Find all large itemsets of size 3

Now count support
for the remaining itemsets

{apple, corn, dates} = 2/4
{corn, dates, tuna} = 2/4

| Transaction | Items |
|---|---|
| T1 | apple, dates, rice, corn |
| T2 | corn, dates, tuna |
| T3 | apple, corn, dates, tuna |
| T4 | corn, tuna |

Minimum support = 50%

Since 2/4 = 50%, both are frequent

# Example for Apriori

## Apriori example: Done!

The whole list of large itemsets for this example is:

{apple}
{corn}
{dates}
{tuna}
{apple, corn}
{apple, dates}
{corn, dates}
{corn, tuna}
{dates, tuna}
{apple, corn, dates}
{corn, dates, tuna}

| Transaction | Items |
|---|---|
| T1 | apple, dates, rice, corn |
| T2 | corn, dates, tuna |
| T3 | apple, corn, dates, tuna |
| T4 | corn, tuna |

Minimum support = 50%

# Algorithm AprioriTid

- Also uses the apriori-gen function
- Generate $\overline{C_k}$ to replace database $D$ for calculating support

$L_1$ = {large 1-itemsets};<br>
$\overline{C}_1$ = database $\mathcal{D}$;<br>
**for** ( $k = 2$; $L_{k-1} \neq \emptyset$; $k$++ ) **do begin**<br>
    $C_k$ = apriori-gen($L_{k-1}$); // New candidates<br>
    $\overline{C}_k = \emptyset$;<br>
    **forall** entries $t \in \overline{C}_{k-1}$ **do begin**<br>
        // determine candidate itemsets in $C_k$ contained<br>
        // in the transaction with identifier $t$.TID<br>
        $C_t = \{c \in C_k \mid (c - c[k]) \in t.\text{set-of-itemsets} \wedge$<br>
            $(c - c[k-1]) \in t.\text{set-of-itemsets}\}$;<br>
        **forall** candidates $c \in C_t$ **do**<br>
            $c$.count++;<br>
        **if** $(C_t \neq \emptyset)$ **then** $\overline{C}_k$ += $< t.\text{TID}, C_t >$;<br>
    **end**<br>
    $L_k = \{c \in C_k \mid c.\text{count} \geq \text{minsup}\}$<br>
**end**<br>
Answer = $\bigcup_k L_k$;

# Example for AprioriTid

- Step 1: generate $L_1$ and $\overline{C_1}$

$L_1$

| Itemset | Support |
|---|---|
| {1} | 2 |
| {2} | 3 |
| {3} | 3 |
| {5} | 3 |

Database

| TID | Items |
|---|---|
| 100 | 1 3 4 |
| 200 | 2 3 5 |
| 300 | 1 2 3 5 |
| 400 | 2 5 |

Min supp=2

$\overline{C}_1$

| TID | Set-of-Itemsets |
|---|---|
| 100 | { {1}, {3}, {4} } |
| 200 | { {2}, {3}, {5} } |
| 300 | { {1}, {2}, {3}, {5} } |
| 400 | { {2}, {5} } |

# Example for AprioriTid

- Step 2.1: generate $C_2$ and $\overline{C_2}$

Database

| TID | Items |
|---|---|
| 100 | 1 3 4 |
| 200 | 2 3 5 |
| 300 | 1 2 3 5 |
| 400 | 2 5 |

Min supp=2

$\overline{C}_1$

| TID | Set-of-Itemsets |
|---|---|
| 100 | { {1}, {3}, {4} } |
| 200 | { {2}, {3}, {5} } |
| 300 | { {1}, {2}, {3}, {5} } |
| 400 | { {2}, {5} } |

$C_2$

| Itemset | Support |
|---|---|
| {1 2} | 1 |
| {1 3} | 2 |
| {1 5} | 1 |
| {2 3} | 2 |
| {2 5} | 3 |
| {3 5} | 2 |

$\overline{C}_2$

| TID | Set-of-Itemsets |
|---|---|
| 100 | { {1 3} } |
| 200 | { {2 3}, {2 5}, {3 5} } |
| 300 | { {1 2}, {1 3}, {1 5}, {2 3}, {2 5}, {3 5} } |
| 400 | { {2 5} } |

# Example for AprioriTid

- Step 2.2: calculate $L_2$

$\overline{C}_2$

| TID | Set-of-Itemsets |
|---|---|
| 100 | { {1 3} } |
| 200 | { {2 3}, {2 5}, {3 5} } |
| 300 | { {1 2}, {1 3}, {1 5}, {2 3}, {2 5}, {3 5} } |
| 400 | { {2 5} } |

$L_2$

| Itemset | Support |
|---|---|
| {1 3} | 2 |
| {2 3} | 2 |
| {2 5} | 3 |
| {3 5} | 2 |

Database

| TID | Items |
|---|---|
| 100 | 1 3 4 |
| 200 | 2 3 5 |
| 300 | 1 2 3 5 |
| 400 | 2 5 |

Min supp=2

# Example for AprioriTid

- Step 3.1: generate $C_3$ and $\overline{C_3}$

$\overline{C}_2$

| TID | Set-of-Itemsets |
|---|---|
| 100 | { {1 3} } |
| 200 | { {2 3}, {2 5}, {3 5} } |
| 300 | { {1 2}, {1 3}, {1 5}, {2 3}, {2 5}, {3 5} } |
| 400 | { {2 5} } |

Database

| TID | Items |
|---|---|
| 100 | 1 3 4 |
| 200 | 2 3 5 |
| 300 | 1 2 3 5 |
| 400 | 2 5 |

Min supp=2

$C_3$

| Itemset | Support |
|---|---|
| {2 3 5} | 2 |

$\overline{C}_3$

| TID | Set-of-Itemsets |
|---|---|
| 200 | { {2 3 5} } |
| 300 | { {2 3 5} } |

# Example for AprioriTid

- 3.2: calculate $L_3$

$\overline{C}_3$

| TID | Set-of-Itemsets |
|---|---|
| 200 | { {2 3 5} } |
| 300 | { {2 3 5} } |

$L_3$

| Itemset | Support |
|---|---|
| {2 3 5} | 2 |

Database

| TID | Items |
|---|---|
| 100 | 1 3 4 |
| 200 | 2 3 5 |
| 300 | 1 2 3 5 |
| 400 | 2 5 |

Min supp=2

# Discussion (In Pairs)

A question from many of you!

What should we consider when evaluating an algorithm to provide a fair and comprehensive comparison with other work?

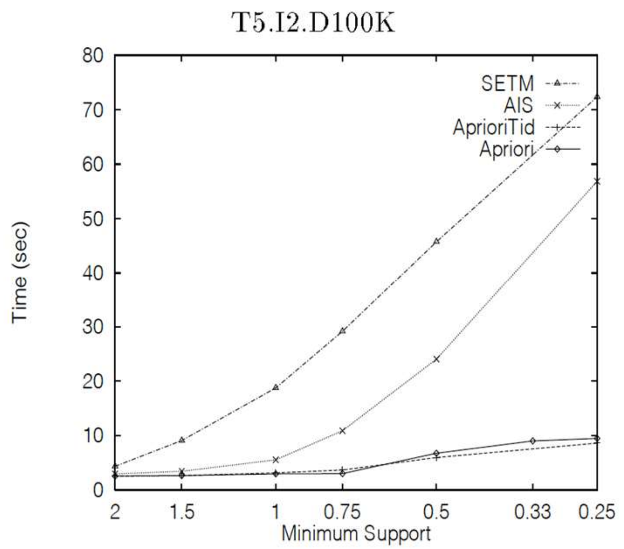
# Performance - dataset

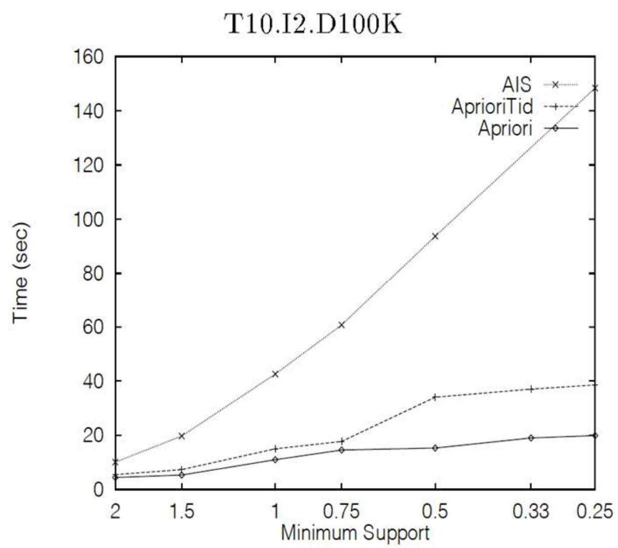
- Generate synthetic transactions to evaluate the performance
- Parameter setting:

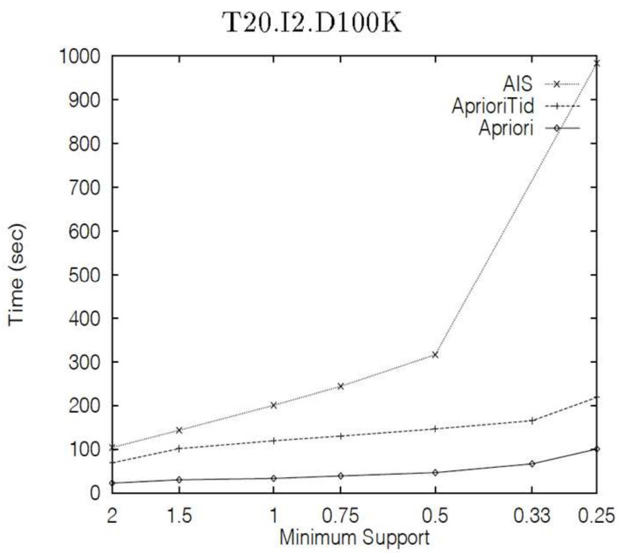
| Name | $\|T\|$ | $\|I\|$ | $\|D\|$ | Size in Megabytes |
|---|---|---|---|---|
| T5.I2.D100K | 5 | 2 | 100K | 2.4 |
| T10.I2.D100K | 10 | 2 | 100K | 4.4 |
| T10.I4.D100K | 10 | 4 | 100K | |
| T20.I2.D100K | 20 | 2 | 100K | 8.4 |
| T20.I4.D100K | 20 | 4 | 100K | |
| T20.I6.D100K | 20 | 6 | 100K | |

# Performance - Execution times

- Minimum support decrease => execution times increase
- Apriori outperforms AIS and SETM
- AprioriTiD is Slower for larger problems



# Performance – per pass

- AprioriTid beats Apriori in later passes
  - The size of $\overline{C_k}$ becomes smaller in later passes

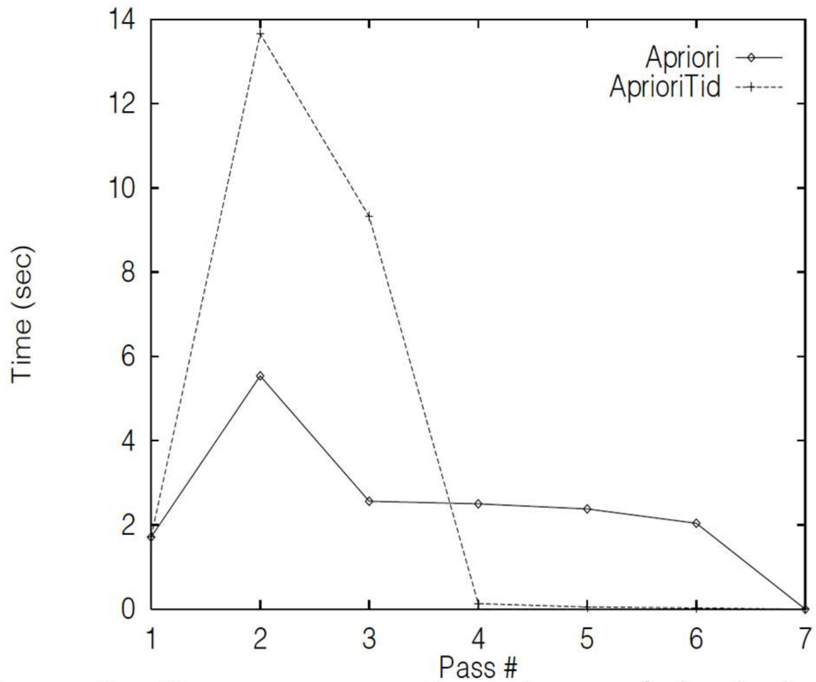
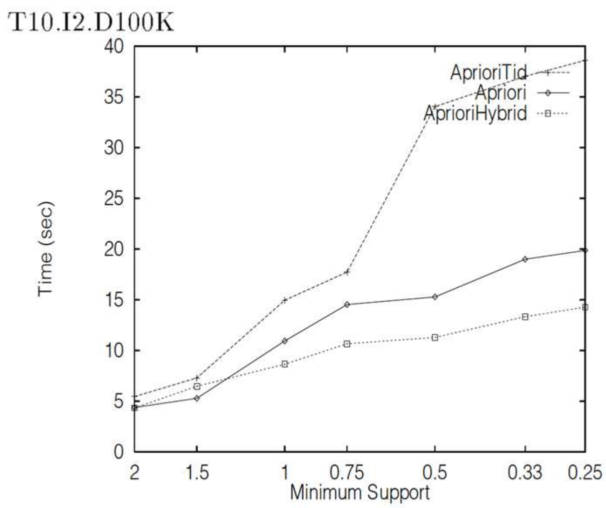
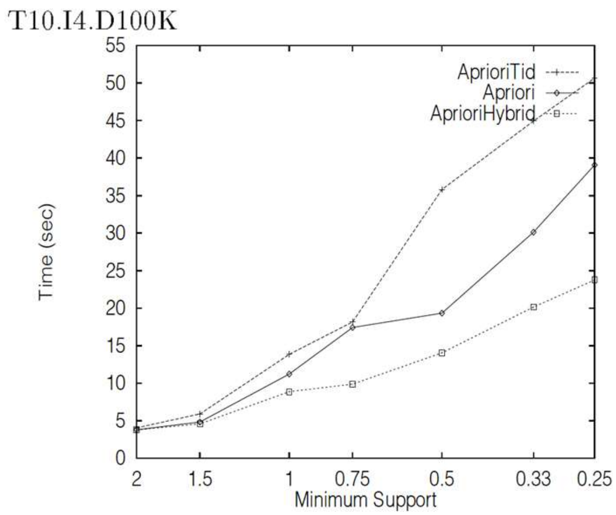


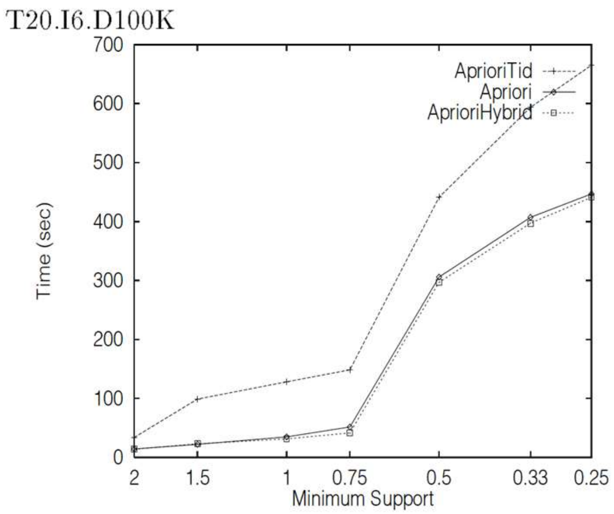

Figure 6: Per pass execution times of Apriori and AprioriTid (T10.I4.D100K, minsup = 0.75%)

# Performance – AprioriHybrid

- Uses Apriori in the initial passes
- Switches to AprioriTid when $\overline{C_k}$ is expected to fit in memory



# Summary

- Two new algorithms, Apriori and AprioriTid are discussed.
- These algorithms outperform AIS and SETM.
- Apriori and AprioriTid can be combined into AprioriHybird.
- AprioriHybrid matches Apriori and AprioriTid when either one wins.

# Discussion (Groups of 3-4)

- The Apriori algorithm was motivated by a very particular business task. But this paper is highly cited (28816 citations in Google Scholar). What do you think are the causes?
  - ❖ Is it the research topic?
  - ❖ The algorithm?
  - ❖ The approach to designing and evaluating a solution?
  - ❖ Something else?

# Bias in OLAP Queries: Detection, Explanation, and Removal
## (Or Think Twice About Your AVG-Query)

Babak Salimi    Johannes Gehrke    Dan Suciu
University of Washington    Microsoft
SIGMOD 2018

Slides adopted from Babak Salimi
Presenter: Yingfeng Lu
Discussion Leader: Jianhao Cao
CPSC 504

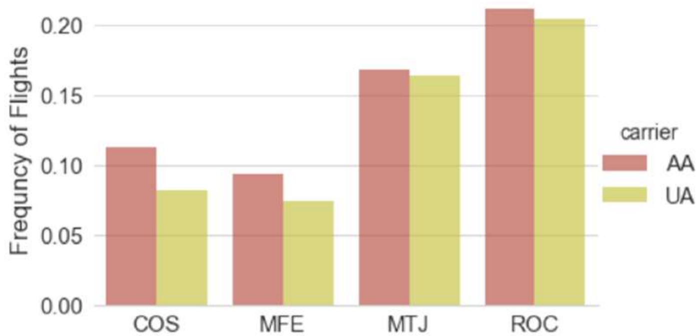
# Biased Queries

- OLAP tools enable complex calculations, analyses, and sophisticated data modeling
- However, inexperienced workers can easily write queries that are biased for decision making
- Answers to OLAP queries can be biased and lead to wrong decisions

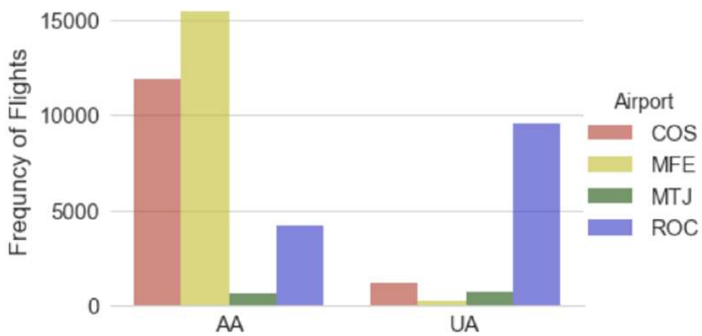
# Example

- Suppose a company wants to choose between the business travel programs offered by *American* (AA) and *United* (UA)

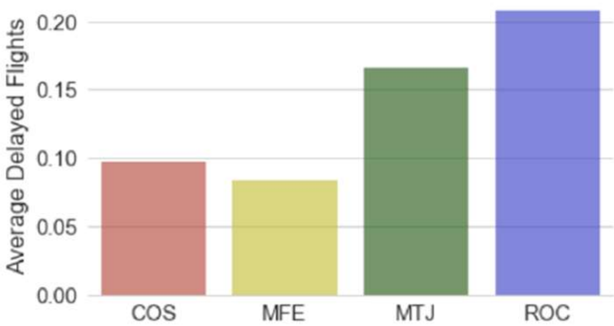
Q: Which airline has a better on-time performance?



Carriers Delay by Airport



Airport by Carrier



Delay by Airport

# Example

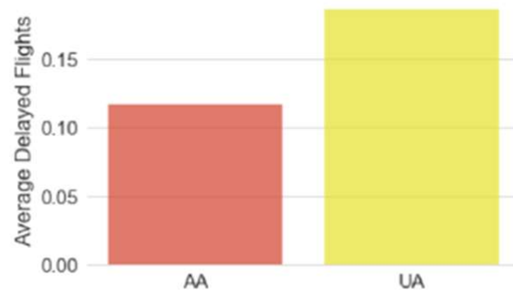
## Q: Which airline has a better on-time performance?

OLAP Query:

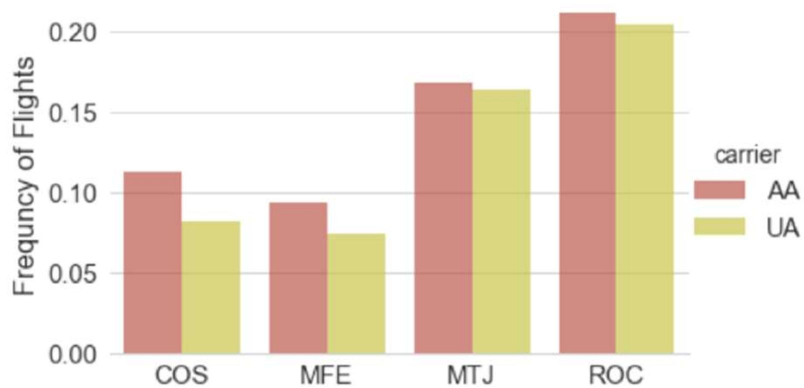
```
SELECT avg(Delayed)
FROM FlightData
GROUP BY Carrier
WHERE Carrier IN ('AA','UA')
  AND Airport IN
   ('COS','MFE','MTJ','ROC')
```

Query Answers:

AA has a lower average flight delay




- Wrong decision because AA has a higher average delay than UA at each of the four airports



# Example

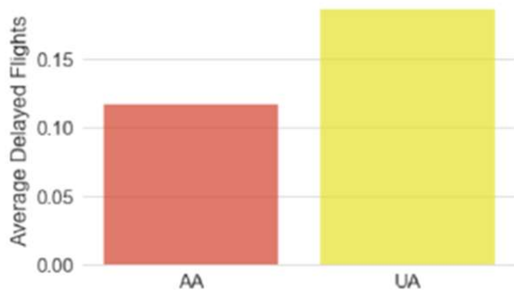
## Q: Which airline has a better on-time performance?

OLAP Query:

```
SELECT avg(Delayed)
FROM FlightData
GROUP BY Carrier
WHERE Carrier IN ('AA','UA')
  AND Airport IN
   ('COS','MFE','MTJ','ROC')
```

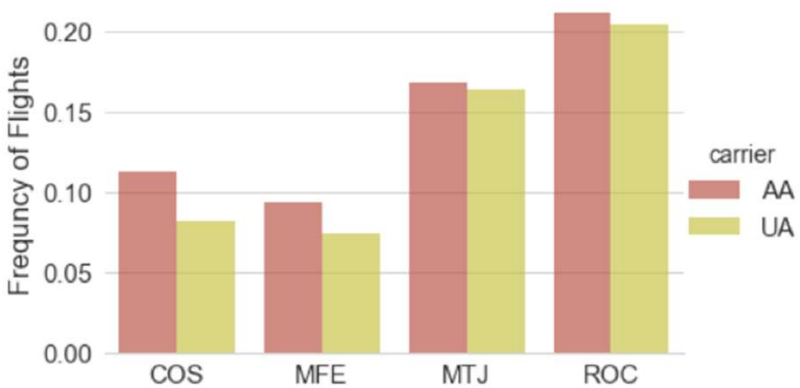
Query Answers:

AA has a lower average flight delay




- Wrong decision because AA has a higher average delay than UA at each of the four airports

**Simpson's paradox**: a trend appears in several groups of data but disappears or reverses when the groups are combined.



# Discussion (In Pairs)

Have you ever seen bias in the data you work on,

or in your research area?

What are the bias examples or causes?

# Causal analysis

- No causal analysis tools exist for OLAP systems.
  - Still use the simplest group-by queries
- Covariate: core of any causal analysis
  - Needs to be controlled to eliminate bias
  - E.g., Airport
- Causal DAG (Directed Acyclic Graph)
  - Edge: a potential cause-effect relationship between attributes
  - Is not given in the paper's setting
  - Inapplicable to compute

# HypDB's novelty

- Covariate discovery: explores only the subset relevant to the current query
- A powerful optimization to significantly speed up the Monte Carlo permutation test
- Explain its findings, and rank the explanations

# Background

**Listing 1: An OLAP query $Q$.**

```
SELECT T,X,avg(Y_1), ... ,avg(Y_e)
FROM D
WHERE C
GROUP BY T,X
```

- set of attributes: $\boldsymbol{X}$
- treatment variable: $T \in \{t_0, t_1\}$
- outcome variable: $Y \in \{0,1\}$
- context for the query $Q$: $\Gamma_i \stackrel{\text{def}}{=} \mathrm{C} \wedge (\mathbf{X} = \mathbf{x}_i)$
- a set of covariates: $\boldsymbol{Z}$

# Biased OLAP Queries – Detecting bias

- Definition of balanced query:
  - The query $Q$ is balanced w.r.t. a set of variables $\boldsymbol{V}$ iff $(T \perp\!\!\!\perp \mathbf{V} | \Gamma_i)$
- Detect biased query by checking $(T \perp\!\!\!\perp \mathbf{V} | \Gamma_i)$

# Biased OLAP Queries – Explaining Bias

Two ways to find ranked explanations for the bias

- Coarse-grained: rank the variables $\boldsymbol{V}$ in terms of their responsibilities for the bias
  - Calculate *Degree of Responsibility* for each $V$
- Fine-Grained: compute the contribution of the triples$(t, y, z)$ that explains the confounding relationships between the ground levels.

## Biased OLAP Queries – Resolving Bias

Rewrite the query to remove the bias

- Partition the data into blocks that are homogeneous on $\boldsymbol{Z}$
- Compute the average of each $\mathrm{Y} \in \boldsymbol{Y}$ Group by $T, X$, in each block
- Aggregate the block's averages by taking their weighted average

# Automatic Covariates Discovery

Given a treatment variable T, compute its parents $\boldsymbol{PA_T}$ in the causal DAG and sets $\boldsymbol{Z} = \boldsymbol{PA_T}$

- Intuition: Z,W are parents of T if T is a common descendant of Z and W

- **CD** algorithm:
  - Phase I: find all parents of T and possibly parents of its children
  - Phase II: keep only parents of T

# Efficient Independence Test

- Monte-Carlo permutation test
  - needs to be performed a sufficiently large number of times
  - requires permuting the entire database
- Permutation test using contingency tables
  - draw random permutations directly from the distribution of all contingency tables with fixed marginals
- Mutual Information Test (MIT): a non-parametric test for significance of conditional mutual information

# Other Optimizations

- Materializing contingency tables
- Caching entropy
- Hybrid independent test

# Experiments

Address four questions:

- Q1: Avoiding false discoveries
  - 50M entries in the FlightData
- Q2: End-to-end results
- Q3: Quality comparison
  - Run on five datasets
- Q4: Efficacy of the optimization techniques
  - need ground truth, generate RandomData

# Experiments

- Q1: Avoiding false discoveries


a) The effect of query rewriting on FlightData.

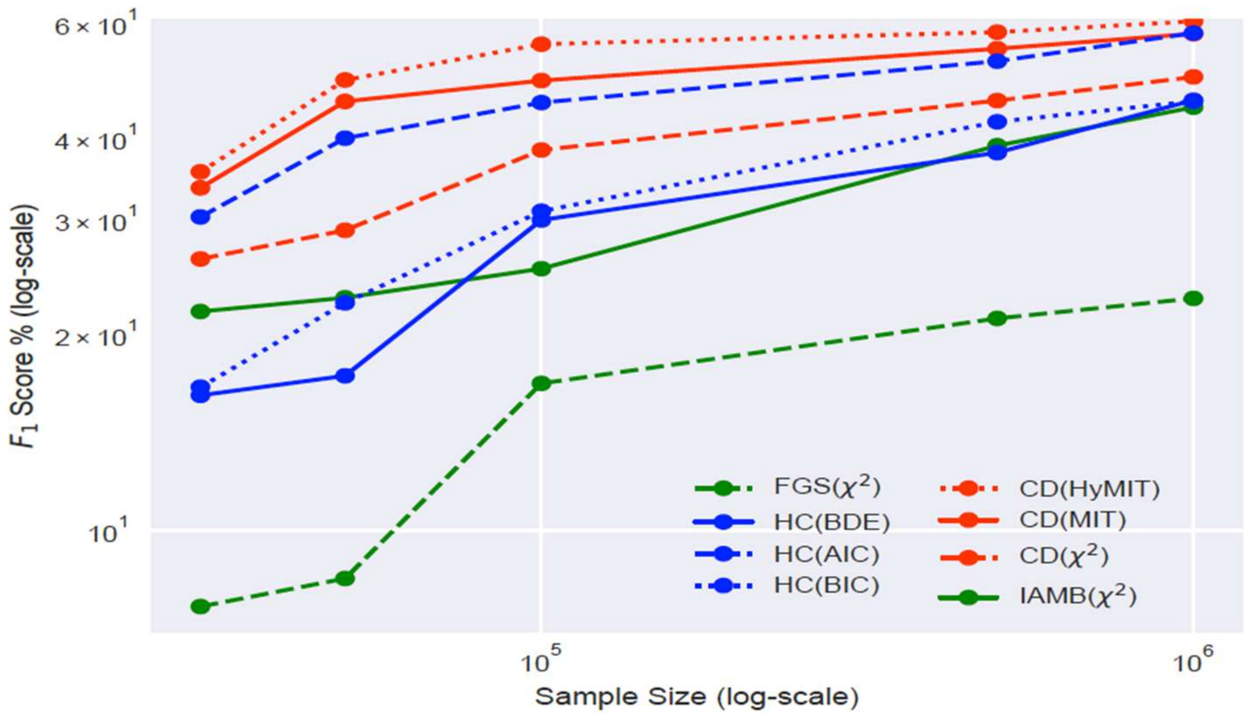
# Experiments

- End-to-end results

| Dataset | Columns [#] | Rows[#] | Det. | Exp. | Res. |
|---|---|---|---|---|---|
| **AdultData** [22] | 15 | 48842 | 65 | <1 | <1 |
| **StaplesData** [49] | 6 | 988871 | 5 | <1 | <1 |
| **BerkeleyData** [3] | 3 | 4428 | 2 | <1 | <1 |
| **CancerData** [15] | 12 | 2000 | <1 | <1 | <1 |
| **FlightData** [42] | 101 | 43853 | 20 | <1 | <1 |

**Table 1: Runtime in seconds for experiments in Sec. 7.3.**

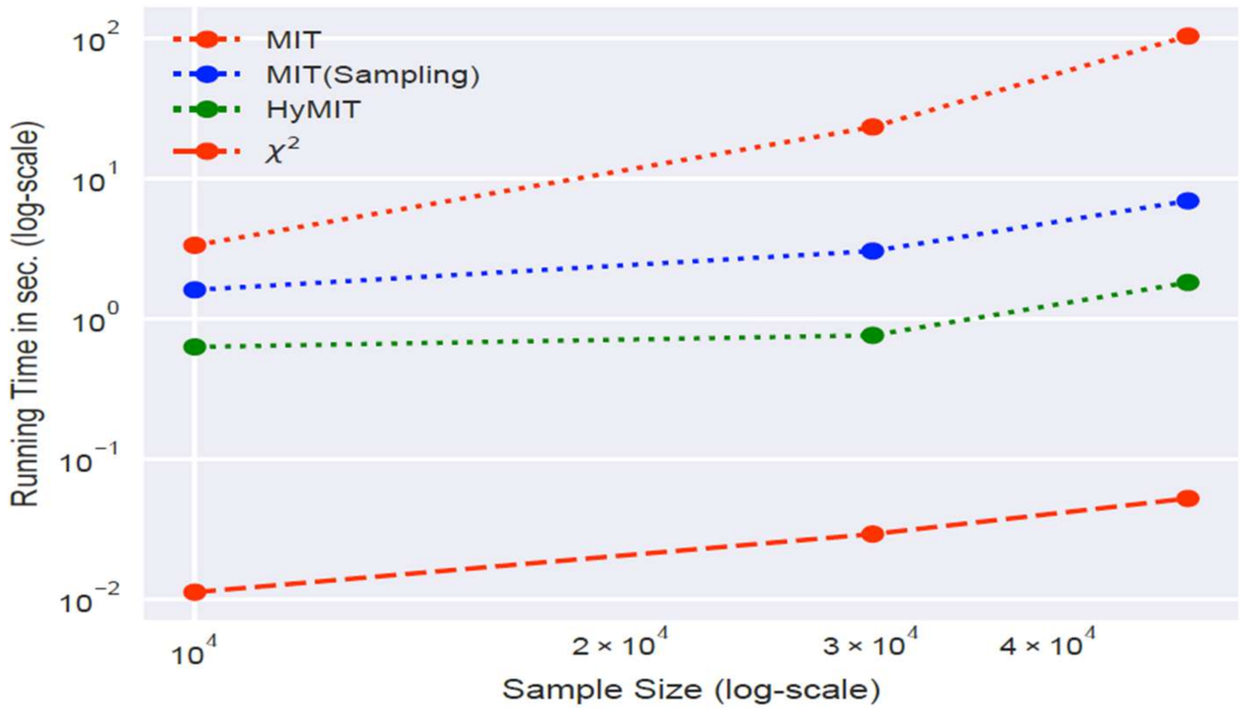
- Quality comparison




**b) Quality comparison.**

# Experiments

- Efficacy of the optimization techniques




**b) Efficacy of the optimizations proposed for independence tests.**

# Summary

- Show that biased queries can be perplexing and lead to statistical anomalies
- Propose HypDB to detect, explain, and to resolve bias in decision-support queries
- Developed an automated method for rewriting the query into an unbiased query

# Discussion (Groups of 4)

As computer scientists, what could we do to help to reduce bias in computer science?

- Pick any perspective that interests you.
  - Academia or industry researchers
  - Software engineers
  - Teaching professors