Today’s Paper(s)

Fast Algorithms for Mining Association Rules
Agrawal and Srikant
VLDB 1994

18,603 citations on Google Scholar

One of the most cited papers in CS
• Acknowledgement:
The following slides have been prepared using several presentations of this paper available on the internet (esp. by Ofer Pasternak and Brian Chase)
Mining Association Rules

• Retailers can collect and store massive amounts of sales data
  – transaction date and list of items

• Association rules:
  – e.g. 98% customers who purchase “tires” and “auto accessories” also get “automotive services” done
  – Customers who buy mustard and ketchup also buy burgers
  – Goal: find these rules from just transactional data (transaction id + list of items)
Applications

• Can be used for
  – marketing program and strategies
  – cross-marketing
  – catalog design
  – add-on sales
  – store layout
  – customer segmentation
Notations

• Items $I = \{i_1, i_2, \ldots, i_m\}$
• $D$: a set of transactions
• Each transaction $T \subseteq I$
  – has an identifier TID
• Association Rule
  – $X \rightarrow Y$
  – $X, Y \subseteq I$
  – $X \cap Y = \emptyset$
Confidence and Support

• Association rule $X \rightarrow Y$

• Confidence $c = \frac{|\text{Tr. with } X \text{ and } Y|}{|\text{Tr. with } X|} \times 100$
  – $c\%$ of transactions in $D$ that contain $X$ also contain $Y$

• Support $s = \frac{|\text{Tr. with } X \text{ and } Y|}{|\text{all Tr.}|} \times 100$
  – $s\%$ of transactions in $D$ contain $X$ and $Y$. 
## Support Example

<table>
<thead>
<tr>
<th>TID</th>
<th>Cereal</th>
<th>Beer</th>
<th>Bread</th>
<th>Bananas</th>
<th>Milk</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>X</td>
<td></td>
<td>X</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>2</td>
<td>X</td>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>3</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>4</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>6</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>7</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
</tbody>
</table>

- **Support(Cereal)**
  - $4/8 = 0.5$
- **Support(Cereal \rightarrow Milk)**
  - $3/8 = 0.375$
# Confidence Example

<table>
<thead>
<tr>
<th>TID</th>
<th>Cereal</th>
<th>Beer</th>
<th>Bread</th>
<th>Bananas</th>
<th>Milk</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>X</td>
<td></td>
<td>X</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>2</td>
<td>X</td>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>3</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>4</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>6</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>7</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>8</td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td></td>
</tr>
</tbody>
</table>

- Confidence(Cereal $\rightarrow$ Milk)
  - $3/4 = .75$
- Confidence(Bananas $\rightarrow$ Bread)
  - $1/3 = .33333...$
Problem Definition

• **Input**
  – a set of transactions D
  – min support (minsups)
  – min confidence (minconf)

• **Goal**
  – generate all association rules that have
    • support >= minsup and
    • confidence >= minconf
X $\rightarrow$ Y is not a Functional Dependency

For functional dependencies
• F.D. = two tuples with the same value of X must have the same value of Y
  – $X \rightarrow Y$ => $XZ \rightarrow Y$ (concatenation)
  – $X \rightarrow Y, Y \rightarrow Z$ => $X \rightarrow Z$ (transitivity)

For association rules
• $X \rightarrow A$ does not mean $XY \rightarrow A$
  – May not have the minimum support
  – Assume one transaction {AX}
• $X \rightarrow A$ and $A \rightarrow Z$ do not mean $X \rightarrow Z$
  – May not have the minimum confidence
  – Assume two transactions {XA}, {AZ}
Divide into two subproblems

1. Find all sets of items (itemsets) that have support above the minimum support
   - #transactions containing them >= threshold
   - these are called large itemsets
2. Use large item sets to find rules with at least minimum confidence
   - Naïve algorithm:
   - For every large itemset p,
   - find all non-empty subsets of p
   - for each such subset q, if support(p)/support(q) >= minconf
   - output q \rightarrow (p - q)

- Paper focuses on subproblem 1
  - if support is low, confidence may not say much
  - subproblem 2 in full version

- Two main algorithms: Apriori and AprioriTID
Determining Large Itemsets

• Algorithms make multiple passes over the data (D) to determine which itemsets are large

• First pass:
  – Count support of individual items
  – Determine which are large

• Subsequent Passes:
  – Use itemsets from previous passes sets to determine new potential” large itemsets (“candidate” large itemsets sets)
  – Count support for candidates from data (D) and remove ones not above minsup to get “actual” large itemsets

• Repeat
Notations

<table>
<thead>
<tr>
<th>$k$-itemset</th>
<th>An itemset having $k$ items.</th>
</tr>
</thead>
<tbody>
<tr>
<td>$L_k$</td>
<td>Set of large $k$-itemsets (those with minimum support). Each member of this set has two fields: i) itemset and ii) support count.</td>
</tr>
<tr>
<td>$C_k$</td>
<td>Set of candidate $k$-itemsets (potentially large itemsets). Each member of this set has two fields: i) itemset and ii) support count.</td>
</tr>
<tr>
<td>$\overline{C}_k$</td>
<td>Set of candidate $k$-itemsets when the TIDs of the generating transactions are kept associated with the candidates.</td>
</tr>
</tbody>
</table>
Algorithm Apriori

\[ L_1 = \{\text{large 1-itemsets}\} \]

For \((k = 2; L_{k-1} \neq \emptyset; k++)\) do begin

\[ C_k = \text{apriori-gen}(L_{k-1}); \]

forall transactions \(t \in D\) do begin

\[ C_t = \text{subset}(C_k, t); \]

forall candidates \(c \in C_t\) do

\[ c.\text{count}++; \]

end

end

\[ L_k = \{c \in C_k | c.\text{count} \geq \text{minsup}\} \]

end

Answer = \(\bigcup_k L_k;\)
Apriori-Gen

- **Join step**
  
  insert into $C_k$
  
  select $p.item_1, p.item_2, p.item_{k-1}, q.item_{k-1}$
  
  from $L_{k-1}p, L_{k-1}q$
  
  where $p.item_1 = q.item_1, \ldots, p.item_{k-2} = q.item_{k-2}, p.item_{k-1} < q.item_{k-1}$

- **Prune step**
  
  forall itemsets $c \in C_k$ do
    
    forall $(k-1)$-subsets $s$ of $c$ do
      
      if $(s \notin L_{k-1})$ then
        
        delete $c$ from $C_k$

  p and q are two large
  
  (k-1)-itemsets identical in all k-2
  
  first items.

  Join by adding the last item of q to p

  Check all the subsets, remove all
  
  candidate with some “small” subset
Apriori-Gen Example - 1

Step 1: Join \((k = 4)\)

Assume numbers 1-5 correspond to individual items

\[
\begin{align*}
L_3 & \quad C_4 \\
\{1,2,3\} & \quad \{1,2,3,4\} \\
\{1,2,4\} & \quad \{1,3,4\} \\
\{1,3,5\} & \quad \{1,3,4\} \\
\{2,3,4\} & \\
\end{align*}
\]
Apriori-Gen Example - 2

Step 1: Join \( k = 4 \)

Assume numbers 1-5 correspond to individual items

\[ L_3 \]
- \{1,2,3\}
- \{1,2,4\}
- \{1,3,4\}
- \{1,3,5\}
- \{2,3,4\}

\[ C_4 \]
- \{1,2,3,4\}
- \{1,3,4,5\}
Apriori-Gen Example - 3

Step 2: Prune ($k = 4$)

- Remove itemsets that can’t have the required support because there is a subset in it which doesn’t have the level of support i.e. not in the previous pass (k-1)

$L_3$
- $\{1,2,3\}$
- $\{1,2,4\}$
- $\{1,3,4\}$
- $\{1,3,5\}$
- $\{2,3,4\}$

$C_4$
- $\{1,2,3,4\}$
- $\{1,3,4,5\}$

No $\{1,4,5\}$ exists in $L_3$
Rules out $\{1, 3, 4, 5\}$
Comparisons with previous algos (AIS, STEM)

$L_{k-1}$ to $C_k$

- Read each transaction $t$
- Find itemsets $p$ in $L_k$ that are in $t$
- Extend $p$ with large items in $t$ and occur later in lexicographic order

$L_3$  
- $\{1,2,3\}$
- $\{1,2,4\}$
- $\{1,3,4\}$
- $\{1,3,5\}$
- $\{2,3,4\}$

$C_4$
- $\{1,2,3,4\}$
- $\{1,2,3,5\}$
- $\{1,2,4,5\}$
- $\{1,3,4,5\}$

$t = \{1, 2, 3, 4, 5\}$

5 candidates compared to 2 in Apriori
Correctness of Apriori

insert into $C_k$
select $p \cdot \text{item}_1, p \cdot \text{item}_2, p \cdot \text{item}_{k-1}, q \cdot \text{item}_{k-1}$
from $L_{k-1}$
$q, \text{item}_1, ..., p \cdot \text{item}_{k-2} = q \cdot \text{item}_{k-2}, p \cdot \text{item}_{k-1} < q \cdot \text{item}_{k-1}$

Show that $C_k \supseteq L_k$

- Any subset of large itemset must also be large
- for each $p$ in $L_k$, it has a subset $q$ in $L_{k-1}$
- We are extending those subsets $q$ in Join with another subset $q'$ of $p$, which must also be large
  - equivalent to extending $L_{k-1}$ with all items and removing those whose $(k-1)$ subsets are not in $L_{k-1}$
- Prune is not deleting anything from $L_k$

forall itemsets $c \in C_k$ do
  forall $(k-1)$-subsets $s$ of $c$ do
    if ($s \notin L_{k-1}$) then
      delete $c$ from $C_k$
Variations of Apriori

• In the k-th pass
  – Not only update $C_k$
  – update candidates $C'_{k+1}$
  – $C'_{k+1} \supseteq C_{k+1}$ since it is generated from $L_k$
  – Can help when the cost of updating and keeping in memory $C'_{k+1} - C_{k+1}$ additional candidates is less than scanning the database
Subset Function

- **Candidate itemsets in** $C_k$ **are stored in a hash-tree (like a B-tree)**
  - interior node = hash table
  - leaf node = itemsets
  - recall that the itemsets are ordered

- **To find all candidates contained in a transaction** $t$
  - if we are at a leaf
    - find which itemsets are contained in $t$
    - add references to them in the answer set
  - if we are at an interior node
    - we have reached it by hashing an item $i$
    - hash on each item that comes after $i$ in $t$
    - reappear
  - if we are at the root, hash on every item in $t$

- **For any itemset** $c$ **in a transaction** $t$
  - the first item must be in the root

\[
L_1 = \{\text{large 1-itemsets}\}
\]

For ($k = 2; L_{k-1} \neq \emptyset ; k + +$) do begin
\[
C_k = \text{apriori- gen}(L_{k-1});
\]
 forall transactions $t \in D$ do begin
\[
C_t = \text{subset}(C_k, t)
\]
forall candidates $c \in C_t$ do
\[
c.\text{count} + +;
\]
end
end

\[
L_k = \{c \in C_k | c.\text{count} \geq \text{minsup}\}
\]
end

\[
\text{Answer} = \bigcup_k L_k;
\]
Problem with Apriori

- Every pass goes over the entire dataset
- Database of transactions is massive
  - Can be millions of transactions added an hour
- Scanning database is expensive
  - In later passes transactions are likely NOT to contain large itemsets
  - Don’t need to check those transactions

\[
L_1 = \{\text{large 1-itemsets}\}
\]

For \( k = 2; L_{k-1} \neq \emptyset; k++ \) do begin
\[
C_k = \text{apriori-gen}(L_{k-1});
\]
forall transactions \( t \in D \) do begin
\[
C_t = \text{subset}(C_k, t)
\]
forall candidates \( c \in C_t \) do
\[
c.\text{count} + +;
\]
end
end
\[
L_k = \{ c \in C_k | c.\text{count} \geq \text{minsup} \}
\]
end

\[
Answer = \bigcup_k L_k;
\]
AprioriTid

- Also uses Apriori-Gen
- But scans the database D only once.
- Builds a storage set $C^*_K$
  - "bar" in the paper instead of *
- Members are of the form < TID, \{X_k\} >
  - each $X_k$ is a potentially large k-itemset present in the transaction TID.
  - For k=1, $C^*_1$ is the database
    - items i as \{i\}
- If a transaction does not have a candidate k-itemset, $C^*_K$ will not contain anything for that TID
- $C^*_K$ may be smaller than #transactions, esp. for large values of k
- For smaller values of k, it may be large
Algorithm AprioriTid

\[ L_1 = \{ \text{large 1-itemsets} \} \]

\[ C_1^\wedge = \text{database } D; \]

For (\( k = 2; L_{k-1} \neq \emptyset ; k ++ \)) do begin

\[ C_k = \text{apriori-gen}(L_{k-1}); \]

\[ C_k^\wedge = \emptyset; \]

forall entries \( t \in C_{k-1} \) do begin

\[ C_t = \{ c \in C_k | (c - c[k] \in t.\text{set-of-items} \]

\[ \wedge (c - c[k-1]) \in t.\text{set-of-items} \}; \]

forall candidates \( c \in C_t \) do

\[ c.\text{count}++; \]

if \((C_t \neq \emptyset)\) then \( C_k^\wedge + = t.TID, C_t \);

end

end

\[ L_k = \{ c \in C_k | c.\text{count} \geq \text{minsup} \} \]

end

Answer = \( \bigcup_{k} L_k; \)

See the examples in the following slides and then come back to the algorithm.
### AprioriTid Example

#### Database

<table>
<thead>
<tr>
<th>TID</th>
<th>Items</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>1 3 4</td>
</tr>
<tr>
<td>200</td>
<td>2 3 5</td>
</tr>
<tr>
<td>300</td>
<td>1 2 3 5</td>
</tr>
<tr>
<td>400</td>
<td>2 5</td>
</tr>
</tbody>
</table>

#### $C_1$

<table>
<thead>
<tr>
<th>TID</th>
<th>Set-of-Itemsets</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>{ {1}, {3}, {4} }</td>
</tr>
<tr>
<td>200</td>
<td>{ {2}, {3}, {5} }</td>
</tr>
<tr>
<td>300</td>
<td>{ {1}, {2}, {3}, {5} }</td>
</tr>
<tr>
<td>400</td>
<td>{ {2}, {5} }</td>
</tr>
</tbody>
</table>

#### $L_1$

<table>
<thead>
<tr>
<th>Itemset</th>
<th>Support</th>
</tr>
</thead>
<tbody>
<tr>
<td>{1}</td>
<td>2</td>
</tr>
<tr>
<td>{2}</td>
<td>3</td>
</tr>
<tr>
<td>{3}</td>
<td>3</td>
</tr>
<tr>
<td>{5}</td>
<td>3</td>
</tr>
</tbody>
</table>

Min support = 2
AprioriTid Example

Min support = 2

Now we need to compute the supports of $C_2$ without looking at the database $D$ from $C_1^*$.
AprioriTid Example

Min support = 2

300 has both \{1\} and \{2\}
Support = 1
also add <300, \{1, 2\}> to C^*_2
AprioriTid Example

Add <100, {1, 3}> to $C^*_2$

Add <300, {1, 3}> to $C^*_2$
### AprioriTid Example

**Min support = 2**

<table>
<thead>
<tr>
<th>$C_2$</th>
<th>Support</th>
</tr>
</thead>
<tbody>
<tr>
<td>{1 2}</td>
<td>1</td>
</tr>
<tr>
<td>{1 3}</td>
<td>2</td>
</tr>
<tr>
<td>{1 5}</td>
<td>1</td>
</tr>
<tr>
<td>{2 3}</td>
<td>2</td>
</tr>
<tr>
<td>{2 5}</td>
<td>3</td>
</tr>
<tr>
<td>{3 5}</td>
<td>2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$\overline{C}_1$</th>
<th>TID</th>
<th>Set-of-Itemsets</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>100</td>
<td>{ {1}, {3}, {4} }</td>
</tr>
<tr>
<td></td>
<td>200</td>
<td>{ {2}, {3}, {5} }</td>
</tr>
<tr>
<td></td>
<td>300</td>
<td>{ {1}, {2}, {3}, {5} }</td>
</tr>
<tr>
<td></td>
<td>400</td>
<td>{ {2}, {5} }</td>
</tr>
</tbody>
</table>

Add the rest
AprioriTid Example

Min support = 2

<table>
<thead>
<tr>
<th>Itemset</th>
<th>Support</th>
</tr>
</thead>
<tbody>
<tr>
<td>{1 2}</td>
<td>1</td>
</tr>
<tr>
<td>{1 3}</td>
<td>2</td>
</tr>
<tr>
<td>{1 5}</td>
<td>1</td>
</tr>
<tr>
<td>{2 3}</td>
<td>2</td>
</tr>
<tr>
<td>{2 5}</td>
<td>3</td>
</tr>
<tr>
<td>{3 5}</td>
<td>2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>TID</th>
<th>Set-of-Itemsets</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>{ {1 3} }</td>
</tr>
<tr>
<td>200</td>
<td>{ {2 3}, {2 5}, {3 5} }</td>
</tr>
<tr>
<td>300</td>
<td>{ {1 2}, {1 3}, {1 5},</td>
</tr>
<tr>
<td></td>
<td>{2 3}, {2 5}, {3 5} }</td>
</tr>
<tr>
<td>400</td>
<td>{ {2 5} }</td>
</tr>
</tbody>
</table>

How C*₂ looks
### AprioriTid Example

Min support = 2

<table>
<thead>
<tr>
<th>Itemset</th>
<th>Support</th>
</tr>
</thead>
<tbody>
<tr>
<td>{1 2}</td>
<td>1</td>
</tr>
<tr>
<td>{1 3}</td>
<td>2</td>
</tr>
<tr>
<td>{1 5}</td>
<td>1</td>
</tr>
<tr>
<td>{2 3}</td>
<td>2</td>
</tr>
<tr>
<td>{2 5}</td>
<td>3</td>
</tr>
<tr>
<td>{3 5}</td>
<td>2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Itemset</th>
<th>Support</th>
</tr>
</thead>
<tbody>
<tr>
<td>{1 3}</td>
<td>2</td>
</tr>
<tr>
<td>{2 3}</td>
<td>2</td>
</tr>
<tr>
<td>{2 5}</td>
<td>3</td>
</tr>
<tr>
<td>{3 5}</td>
<td>2</td>
</tr>
</tbody>
</table>

The supports are in place
Can compute \(L_2\) from \(C_2\)
AprioriTid Example

Min support = 2

Next step
**AprioriTid Example**

Min support = 2

<table>
<thead>
<tr>
<th>Itemset</th>
<th>Support</th>
</tr>
</thead>
<tbody>
<tr>
<td>{2, 3, 5}</td>
<td></td>
</tr>
</tbody>
</table>

Look for transactions containing \{2, 3\} and \{2, 5\}

Add \(<200, \{2,3,5\}>\) and \(<300, \{2,3,5\}>\) to \(C_3^*\)

<table>
<thead>
<tr>
<th>TID</th>
<th>Set-of-Itemsets</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>{ {1 3} }</td>
</tr>
<tr>
<td>200</td>
<td>{ {2 3}, {2 5}, {3 5} }</td>
</tr>
<tr>
<td>300</td>
<td>{ {1 2}, {1 3}, {1 5}, {2 3}, {2 5}, {3 5} }</td>
</tr>
<tr>
<td>400</td>
<td>{ {2 5} }</td>
</tr>
</tbody>
</table>

```plaintext
forall entries \(t \in \overline{C}_{k-1}\) do begin
  // determine candidate itemsets in \(C_k\) contained
  // in the transaction with identifier \(t.\text{TID}\)
  \(C_t = \{c \in C_k \mid (c - c[k]) \in t.\text{set-of-itemsets} \land
  (c - c[k-1]) \in t.\text{set-of-itemsets}\};\)
  forall candidates \(c \in C_t\) do
    \(c.\text{count}++;\)
    if \((C_t \neq \emptyset)\) then \(\overline{C}_k \text{ += } \langle t.\text{TID}, C_t \rangle;\)
end
```
AprioriTid Example

Min support = 2

C_3 has only two transactions (we started with 4)
L_3 has the largest itemset
C_4 is empty
Stop
Discovering Rules
(from the full version of the paper)

Naïve algorithm:

• For every large itemset p
  – Find all non-empty subsets of p
  – For every subset q
    • Produce rule q \( \rightarrow \) (p-q)
    • Accept if \( \frac{\text{support}(p)}{\text{support}(q)} \geq \text{minconf} \)
Checking the subsets

- For efficiency, generate subsets using recursive DFS. If a subset $q$ does not produce a rule, we do not need to check for subsets of $q$.

Example

Given itemset: $ABCD$
If $ABC \rightarrow D$ does not have enough confidence
then $AB \rightarrow CD$ does not hold
Reason

For any subset q’ of q:

\[ \text{Support}(q') \geq \text{support}(q) \]

\[ \text{confidence} \ (q' \rightarrow (p-q')) \]
\[ = \ \frac{\text{support}(p)}{\text{support}(q')} \]
\[ \leq \ \frac{\text{support}(p)}{\text{support}(q)} \]
\[ = \ \text{confidence} \ (q \rightarrow (p-q)) \]
forall large itemsets $l_k, k \geq 2$ do
  $\text{genrules}(l_k, l_k)$

procedure genrules($l_k$):large k-itemset, $a_m$: large m-itemset

$A = \{(m-1)$-itemset $a_{m-1} | a_{m-1} \subseteq a_m\}$;
for all $a_{m-1} \in A$ do begin
  $conf = \text{support}(l_k)/\text{support}(a_{m-1})$
  if ($conf \geq \text{minconf}$) then begin
    output the rule $a_{m-1} \Rightarrow (l_k - a_{m-1})$
    if ($m-1 > 1$) then
      call genrules($l_k, a_{m-1}$);
  end
end

Check all the large itemsets
Check all the subsets
Check confidence of new rule
Output the rule
Continue the DFS over the subsets.
If there is no confidence the DFS branch cuts here
Faster Algorithm

• If \((p-q) \rightarrow q\) holds than all the rules 
  \((p-q') \rightarrow q'\) must hold
  – where \(q' \subseteq q\) and is non-empty

Example:
If \(AB \rightarrow CD\) holds,
then so do \(ABC \rightarrow D\) and \(ABD \rightarrow C\)

Idea
• Start with 1-item consequent and generate larger consequents
• If a consequent does not hold, do not look for bigger ones
• The candidate set will be a subset of the simple algorithm
Performance

• Synthetic data modeling “real world”
  – People tend to buy things in sets
• Used the following parameters:

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>( D )</td>
<td>Number of transactions</td>
</tr>
<tr>
<td>( T )</td>
<td>Average size of the transactions</td>
</tr>
<tr>
<td>( I )</td>
<td>Average size of the maximal potentially large itemsets</td>
</tr>
<tr>
<td>( L )</td>
<td>Number of maximal potentially large itemsets</td>
</tr>
<tr>
<td>( N )</td>
<td>Number of items</td>
</tr>
</tbody>
</table>

• The above are used in the names of the datasets: **T10I2D100K**
• Pick the size of the next transaction from a Poisson distribution with mean | \( T \) |
• Randomly pick determined large itemset and put in transaction, if too big overflow into next transaction
Performance

• Support decreases => time increases
• Apriori beats AIS and SETM
  – their candidate set is much larger
• AprioriTID is “almost” as good as Apriori, BUT Slower for larger problems
• \( C_k^* \) does not fit in memory and increases with #transactions
Performance

• AprioriTid is effective in later passes
  – Scans $C^*_k$ instead of the original dataset
  – becomes small compared to original dataset

• When fits in memory, AprioriTid is faster than Apriori
AprioriHybrid

• Use Apriori in initial passes
• Switch to AprioriTid when it can fit in memory
  – estimate the size of $C^*_k$ if it had been generated
  – $= \sum_{c \in C_k} \text{support}(c) + \#\text{transactions}$
  – if it fits in memory and fewer larger candidates in the current pass than previous pass, then switch
  – to avoid the case that $C^*_k$ fits in the current pass but not in the next pass

• Switch happens at the end of the pass
  – Has some overhead to switch

• Relies on size drop
  – If switch happens late, will have slightly worse performance

• Still mostly better or as good as apriori
Summary

• Association rules are important
• This paper gives algorithms to find all association rules with required support and confidence
• Perform better than previous algorithms
• Scale well for large datasets