Geometrically Inspired Itemset Mining*

*Florian Verhein, Sanjay Chawla
IEEE ICDM 2006

Outline

- **Introduction**
- Item Enumeration, Row Enumeration or ?
- Theoretical Framework
- Data Structure
- Algorithm – GLIMIT
- Complexity
- Evaluation

- FIM is the most time consuming part in ARM.
- Traditionally, we use item enumeration type algorithms to mine the dataset for FIM.
- Multiple passes of the original dataset.
- Elements:

  T: transaction set, each transaction t ∈ T

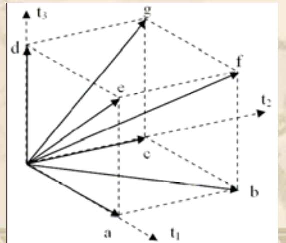
  I: a set that contains all the items. t I

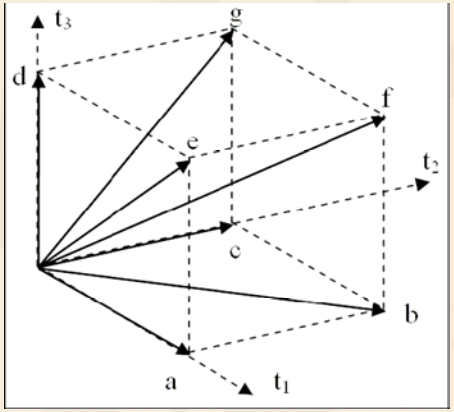
  FIM: an itemset i T and σ (i) ⩾ minSup

- Transpose the original dataset:
- For each row, x_i, contains transactions containing i. $x_i = \{ t.tid : t \in T \wedge i \in t \}$.
- Here, we call x_i an **itemvector.**

  i.e, it represent an item in the space spanned by the transactions.

Traditional	Transposed
$t_1 : 1, 2, 5$	$1 : t_1, t_2$
$t_2 : 1, 2, 3, 4$	$2 : t_1, t_2, t_3$
$t_3 : 2, 4, 5$	$3 : t_2$
	$4 : t_2, t_3$
	$5 : t_1, t_3$





label	corresponding itemsets
a	{1,5}
b	{1},{1,2}
c	{3}, {1,3}, {1,4}, {2,3}, {3,4}, {1,2,3}, {1,2,4}, {1,3,4}, {2,3,4}, {1,2,3,4}
d	{4,5}
e	{5},{2,5}
f	{2}
g	{4},{2,4}

An itemset I' I can also be represented as an itemvector.

$x_{I'} = \{ t.tid : t \in T \wedge I' \quad t \}$

Exmaple: { 2,4 }

$x_4 = \{ t_2,t_3 \}$ is at g.

$x_2 = \{t_1,t_2,t_3 \}$ is at f. So:

$x_{\{2,4\}} = x_2 \cap x_4 = \{ t_2,t_3 \}$ is at g.

Note: $\sigma (x_{I'}) = |x_{I'}|$

- Three key points:
  (1) An item or itemset can be represented by a vector.
  (2) Create vectors that represent itemsets by performing operations on the item -vector. (e.g. intersect itemvectors)
  (3) We can evaluate a measure by using a certain function on the itemvectors. (e.g. Size of an itemvector can be considered as the support of the itemset.)
  ※ These three points can be abstracted to two functions and one operator.(g(),f(),o)

- Preliminary illustration
- For simplicity, we instantiate g(), f() and o for traditional FIM. Bottom-up scanning in transposed dataset row by row. (minSup = 1)
- Check x_5 and x_4, {4} and {5} are frequent.
- $x_{\{4,5\}} = x_4 \cap x_5 = \{t_3\}$
- {3} is frequent
- $x_{\{3,5\}} = x_3 \cap x_5 =$
- $x_{\{3,4\}} = x_3 \cap x_4 = \{t_2\}$
- {3,4,5} is not frequent.
- Continue with x_2

Traditional	Transposed
$t_1 : 1,2,5$	$1 : t_1, t_2$
$t_2 : 1,2,3,4$	$2 : t_1, t_2, t_3$
$t_3 : 2,4,5$	$3 : t_2$
	$4 : t_2, t_3$
	$5 : t_1, t_3$

- A single pass generate all frequent itemsets.
- After processing n itemvectors corresponding to items in {1,2,3…n}, any itemset L {1,2,3…n} will have been generated.
- Transposed format and itemvector allow all these to work.

- **Problem:**
- Space:
- Itemvectors take up significant space (as many as frequent itemsets, worst: $2^{|I|}-1$)
- Time:
- Recomputation. (Not linear, actually exponential)
- Example: $x_{\{1,2,3\}}$ is created, when $x_{\{1,2,3,4\}}$ is needed, we want to use $x_{\{1,2,3\}}$ to compute it rather than recalculate $x_1 \cap x_2 \cap x_3 \cap x_4$.
- Challenge:
- use little space while avoid re-computation.

- **GLIMIT** (Geometrically Inspired Linear Itemset Mining In the Transpose.)
- Using time roughly linear to the number of itemset.
- At worst using n'+ L/2 , n' denote the number of 1-frequent itemset, L is the length of the longest frequent itemset.
- Based on these facts and the geometric inspiration of itemvector.

- Linear space and linear time.
- One pass without candidate generation.
- Based on itemvector framework.

Sounds pretty nice!

Outline

- Introduction
- **Item Enumeration, Row Enumeration or ?**
- Theoretical Framework
- Data Structure
- Algorithm – GLIMIT
- Complexity
- Evaluation

▲Item enumeration. Can prune effectively

Based on anti-monotonic property.

Apriori-like, effective When |T|>>|I|

▲Row enumeration.

Intersect transactions (row based).

Need to keep transposed table for counting purpose.

Effective When |T|<<|I|

- GLIMIT
- Hard to define what it really belongs to?
- Need to keep the transposed table
- Intersect itemvectors in the transposed table rather than intersect transections.
- Search through the itemset space but scan original dataset column-wise. Transpose has never been considered by previous item enumeration approach.
- Conclusion: it is still an item enumeration method.

Outline

- Introduction
- Item Enumeration, Row Enumeration or ?
- **Theoretical Framework**
- Data Structure
- Algorithm – GLIMIT
- Complexity
- Evaluation

- Previously, we consider itemvectors as sets of transactions and perform intersection among them to generate longer itemsets. Also, cardinality function are used to evaluate the support of an itemvector.
- Now, abstract these operations.
- (Recall: $x_{I'}$: itemvector for I', or, set of transactions that contain I')

- Suppose X is the space spanned by all $x_{I'}$
- We have:
- **Definition 1** $g : X \rightarrow Y$ is a transformation on the original itemvector to a different representation $y_{I'} = g(x_{I'})$ in a new space Y .
- The output is still an itemvector.

- **Definition 2** $\circ$ is an operator on the transformed itemvectors so that

$$y_{I' \cup I''} = y_{I'} \circ y_{I''} = y_{I''} \circ y_{I'}$$

- **Definition 3** $f : Y \rightarrow R$ is a measure on itemsets, evaluated on transformed itemvectors. We write $m_{I'} = f(y_{I'})$.

- Definition:
- Suppose $I' = \{i_1, ..., i_q\}$:
- $Interestingness(I') = f(g(x_{i1}) \circ g(x_{i2}) \circ ... \circ g(x_{iq}))$
- So for an interesting measure, we need to find the appropriate g(),$\circ$, f().
- For this presentation, we specifically consider support of an itemset, so the calculation can be represented using the above definitions as:

- g( ): bit-wise transformation
- $\circ$: Intersection $\cap$, bitwise AND
- f( ) : | | or sum( )
- Example:
- Itemvectors: $x_1 = \{t_1, t_2\}$, $x_2 = \{t_1, t_3\}$
- $y_1 = g(x_1) = 110$, $y_2 = g(x_2) = 101$
- $y_1 \circ y_2 = y_1 \cap y_2 = 100 = y_{\{1,2\}}$
- $f = sum(y_{\{1,2\}}) = 1 = y_1 \cdot y_2$
- $\sigma(\{1,2\}) = 1$
- Actually dot product

- Obviously, different definitions of g(),∘, f(). applies to different measures.
- Definition: (not needed for support)
- F : $R^k \rightarrow R$ is a measure on an itemset I' that supports any composition of measures (provided by f(·)) on any number of subsets of I'. That is, $M_{I'} = F(m_{I'_1}, m_{I'_2}, \ldots m_{I'_k})$, where, $m_{I'_i} = f(y_{I'_i})$, and $I_1', I_2' \ldots I_k'$ are k arbitrary subsets of I'.
- Interestingness (I') = $F(m_{I'_1}, m_{I'_2}, \ldots m_{I'_k})$

- If k =1, F( ) = f( ). Support computation function
- Example: (part of spatial colocation mining)

  The minPI of an itemset I' = {1, ..., q} is

  $minPI(I') = min_i\{ \sigma (I') / \sigma (\{i\}) \}$. Suppose

  $m_{I'} = \sigma (I')$. g( ), ∘ , f() are defined the same as before.

Outline

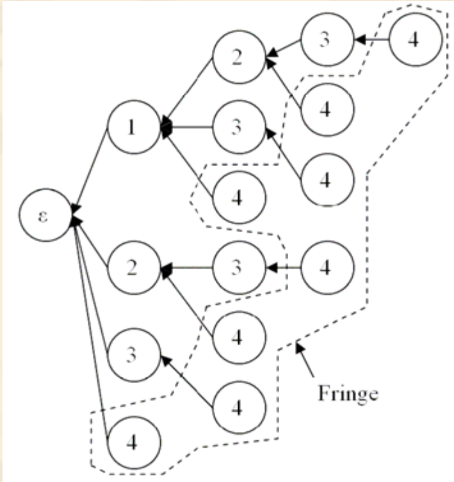
- Introduction
- Item Enumeration, Row Enumeration or ?
- Theoretical Framework
- **Data Structure**
- Algorithm – GLIMIT
- Complexity
- Evaluation

- Structures for GLIMIT:
- Prefix tree:
  - Store itemset I' as s sequence $\langle i_1, i_2 \ldots \ldots i_k \rangle$,

    The order of the item is fixed. (an itemvector)

    Each node of the tree represent a sequence.

    (A prefixNode)
  - itemset = itemvector = sequence = prefixNode
  - PrefixNode tuple = (parent, depth, m(M), item
  - How to recover a sequence?



- Fringe contains maximal itemsets. (for ARM)

Outline

- Introduction
- Item Enumeration, Row Enumeration or ?
- Theoretical Framework
- Data Structure
- **Algorithm – GLIMIT**
- Complexity
- Evaluation

GLIMIT

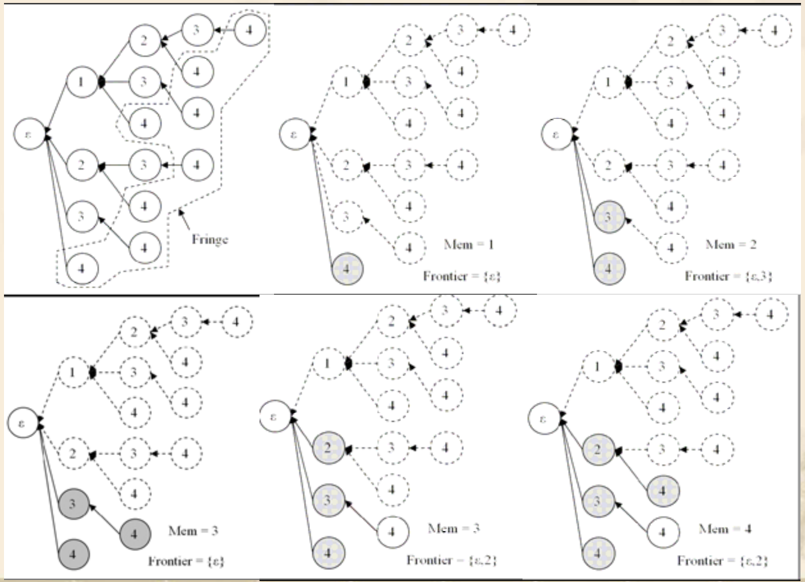
- Depth first search and bottom up scanning.
- 5 facts to help save space and avoid re-computation (time).

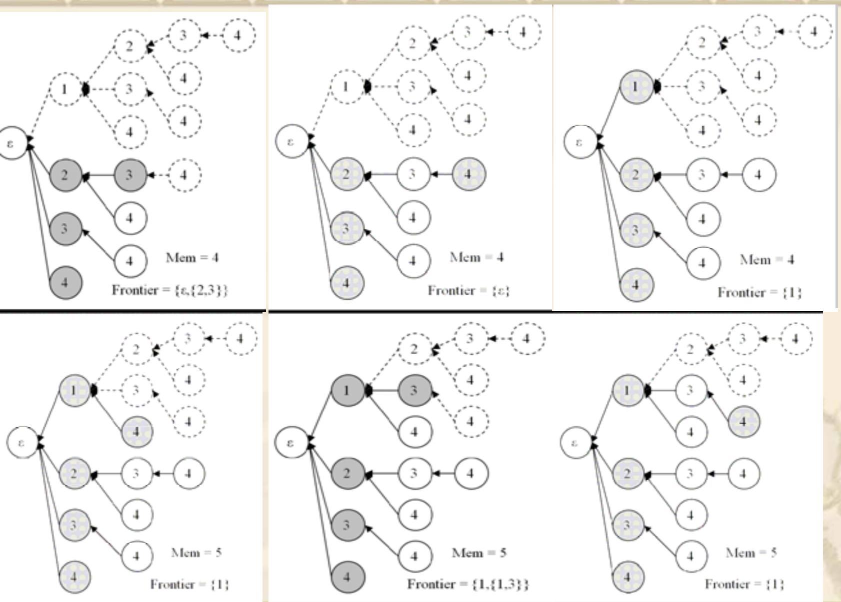
- Fact1:
- Incrementally apply rule $y_{I' \cup \{i\}} = y_{I'} \circ y_i$ i.e, we only have to keep a single itemvector in memory when generating a child of a node.
- Note, for root, we have to keep all the single itemvectors which represent the root's child.

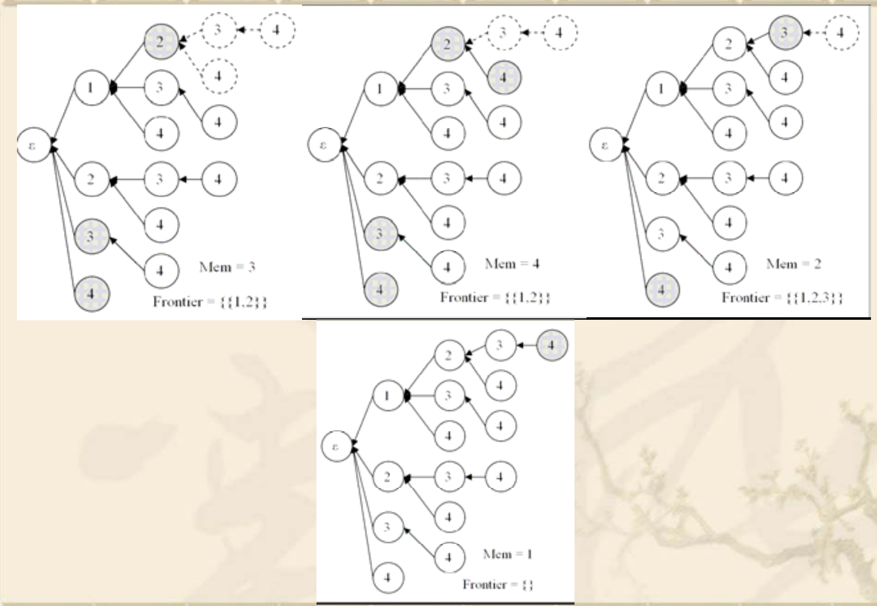
- Fact2: we only expand nodes which have one or more siblings below it. i.e. we check $< i_a, i_b, \ldots, i_i, i_j, i_k>$ only if siblings $< i_a, i_b, \ldots, i_i, i_j >$ and $< i_a, i_b, \ldots, i_i, i_k>$ are in the prefix tree. Here, $k > j$

- Fact3: we use the depth first procedure, when a PrefixNode p is created, then all PrefixNodes corresponding to the subsets of p's itemset will already have been generated.

- Fact4: If PrefixNode (depth>1) have no children to expand, its itemvector will be abandoned. (Note apply for node with depth>1)
- Fact5: when a topmost child of node p is created or checked, delete the itemvector of p. (Note: apply for node with depth>1)
- Fact6: If we create a PrefixNode p on the top- most branch, suppose p stands for $<i_1, i_2, ..., i_k>$, then itemvectors for the any single item in p can be deleted.



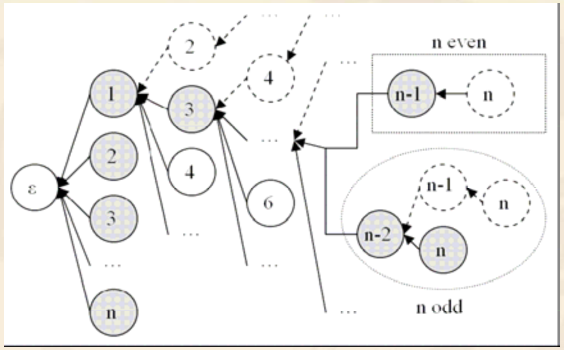




Outline

- Introduction
- Item Enumeration, Row Enumeration or ?
- Theoretical Framework
- Data Structure
- Algorithm – GLIMIT
- **Complexity**
- Evaluation

- Time: roughly linear in the number of frequent itemsets.(Avoid recomputation) Building and mining happen simultaneously.
- Space: we only consider itemvectors needed to save in memory.
- Need to keep all itemvectors for single items until reaching the top-most.
- Need to keep the itemvector for a node if not all children of it has been checked.
- Now consider the worst case:



- Suppose all itemsets are freqent.
- n itemvectors for single items. n/2 for the nodes on the path. (They are not fully expanded.)
- So, worst case is n + n/2 − 1

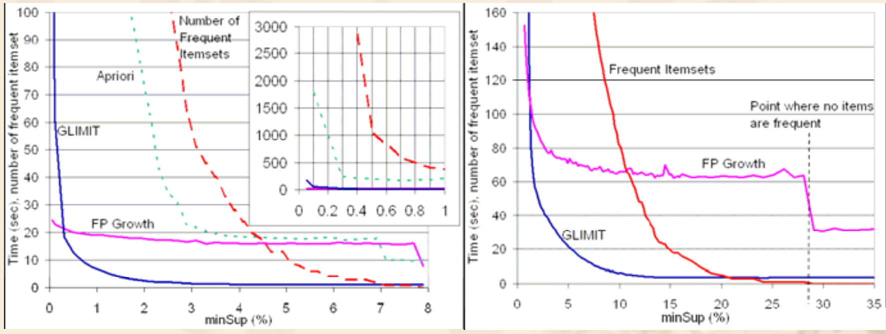
- A closer bound:

  Let n be the number of items, and n’ $\leqslant$ n be the number of frequent items. Let L $\leqslant$ n’ be the size of the largest itemset. GLIMIT uses at most n’ + L/2 − 1 itemvectors of space.
- Much better in practical situation.
- Bottom up
- Depth first from left to right

Outline

- Introduction
- Item Enumeration, Row Enumeration or ?
- Theoretical Framework
- Data Structure
- Algorithm – GLIMIT
- Complexity
- **Evaluation**

- Two datasets with 100,000 transactions each
- Contain 870 and 942 items respectively.



- When MinSup > a certain threshold. GLIMIT outperforms FPGrowth.
- Reason:

For FPGrowth:

● Build the tree and then conditional pattern

● Mine conditional FP-tree iteratively.

(Search by following the links in the tree.)

● It pays off if the minsupport is very small. But if minsupport is big, then space and time are wasted.)

- For GLIMIT:

● Use time and space as needed.

● One pass without generation, linear time and space.

● No resource-consuming mining procedures

● Beaten by FP Growth when MinSup is small because too many bitwise operation decrease the overall efficiency.

Last but not least…

- GLIMIT is somewhat trivial in this paper.
- What is the main purpose?

★ Itemvectors in transaction space

★ A framework for operating on itemvectors

(Great flexibility in selecting measures and transformations on original data)

★ New class of algorithms. Glimit is an instantiation of the concepts.

★ Future work: Geometric inspired measures and transformations for itemset mining.

Thanks

Q?

Nov 29th