

An Introduction to Frequent Itemset Mining using GLIMIT

Florian Verhein
fverhein@it.usyd.edu.au

School of Information Technologies,
The University of Sydney,
Australia

Copyright 2008 Florian Verhein.

January 10, 2008



Outline

Introduction

- Basic Concepts

- Frequent Itemset Mining and GLIMIT

The Geometry of Itemsets

Algorithm

- Introduction

- Challenges

- Data Structure

- GLIMIT

- Example

Performance



Basic Concepts Review

- ▶ Set of *items* $I = \{i_1, i_2, \dots, i_m\}$
 - ▶ **Example:** $I = \{1, 2, 3, 4, 5\}$
- ▶ Set of *transactions* $T = \{t_1, t_2, \dots, t_n\}$, where each $t_i \subseteq I$
 - ▶ **Example:** $T = \{t_1, t_2, t_3\}$ where

Transaction ID	Transaction
t_1	$\{1, 2, 5\}$
t_2	$\{1, 2, 3, 4\}$
t_3	$\{2, 4, 5\}$

- ▶ The *support* of an *itemset* $I' \subseteq I$ is the number of transactions containing I' :
 - ▶ $support(I') = |\{t \in T : I' \subseteq t\}|$



Frequent Itemset Mining (FIM) and GLIMIT

- ▶ **FIM Problem Definition:** Find all itemsets $I' \subseteq I$ so that $support(I') \geq minSup$ – i.e.: find all “Frequent Itemsets”.
- ▶ **FIM Motivation:** The most computationally expensive task in *Association Rule Mining*.
- ▶ **GLIMIT** is an efficient itemset mining algorithm that is based on a geometric interpretation of itemsets.
 - ▶ GLIMIT operates very differently to Apriori or FP-Growth.
 - ▶ operates using functions and operators on vectors, rather than counting.
 - ▶ It makes a *single* pass over the *transposed* data-set,
 - ▶ does not perform *candidate generation* and
 - ▶ does not pre-build a compressed data structure such as the FP-Tree.
 - ▶ It outperforms Apriori, and outperforms FP-Growth above a (typically small) support threshold.
 - ▶ Run-time performance is *linear* in the itemsets examined, and space is *linear* in the size of the data-set.



Itemset-vectors

- ▶ Can calculate support of itemsets using operations and functions on *itemset-vectors*.

Example:

- ▶ $x_{\{2,4\}} = x_{\{2\}} \cap x_{\{4\}} = \{t_1, t_2, t_3\} \cap \{t_2, t_3\} = \{t_2, t_3\}$
(operator \cap)
- ▶ $support(\{2, 4\}) = |x_{\{2,4\}}| = |\{t_2, t_3\}|$ (function $|\cdot|$)

GLIMIT operates exclusively using functions and operators on *itemset-vectors*.

- ▶ Can implement itemset-vectors using *bit-vectors* (*fast*).
 - ▶ Bit is set if corresponding transaction contains the itemset.

Example:

- ▶ $x_{\{2\}} = 111$ (order of bits: t_1, t_2, t_3)
- ▶ $x_{\{4\}} = 011$
- ▶ $x_{\{2,4\}} = 111 \text{ AND } 011 = 011$
- ▶ $support(\{2, 4\}) = setBits(011) = 2$



Introduction to GLIMIT

Transposed Data-set:

Item i	$x_{\{i\}}$
1	$\{t_1, t_2\}$
2	$\{t_1, t_2, t_3\}$
3	$\{t_2\}$
4	$\{t_2, t_3\}$
5	$\{t_1, t_3\}$

Example: Assume $minSup = 2$.

1. Read $x_{\{1\}} = \{t_1, t_2\}$. $support(x_{\{1\}}) = 2$ so $\{1\}$ is frequent.
2. Read $x_{\{2\}} = \{t_1, t_2, t_3\}$. $support(x_{\{2\}}) = 3$ so $\{2\}$ is frequent.
 - 2.1 Since both $x_{\{1\}}$ and $x_{\{2\}}$ are now available, can create $x_{\{2,1\}} = x_{\{2\}} \cap x_{\{1\}} = \{t_1, t_2\}$. $\{1, 2\}$ is frequent.
3. Read $x_{\{3\}} = \{t_2\}$. $support(x_{\{3\}}) = 1$. $\{3\}$ is NOT frequent. No point considering $\{3, 1\}$, $\{3, 2\}$, etc.
4. Read $x_{\{4\}} = \{t_2, t_3\}$. $\{4\}$ is frequent.
 - 4.1 $x_{\{4,1\}} = x_{\{4\}} \cap x_{\{1\}} = \{t_2\}$ so $\{4, 1\}$ is NOT frequent.
 - 4.2 $x_{\{4,2\}} = x_{\{4\}} \cap x_{\{2\}} = \{t_2, t_3\}$ so $\{4, 2\}$ is frequent. Since $\{4, 1\}$ is not frequent, no point considering $\{4, 2, 1\}$.

Since $\{3\}$ is not frequent, no point considering $\{4, 3\}$ or any supersets.

5. Read $x_{\{5\}} = \{t_1, t_3\}$. $\{5\}$ is frequent. etc...

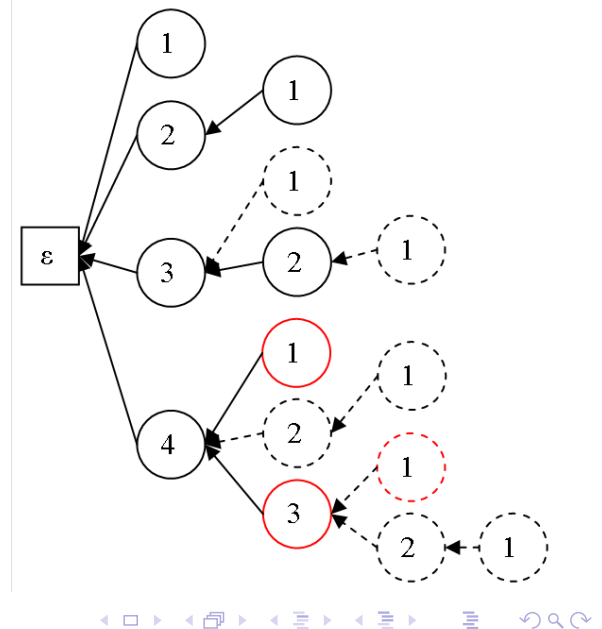


GLIMIT

- ▶ Makes a single pass over the *transposed* data-set, reading in one line (*item-vector*) at a time.
- ▶ Operates in a *strict depth first* fashion (“traversal over *PrefixTree*”)
- ▶ *Automatically* (i.e for free) makes partial use of *support’s anti-monotonicity* (i.e. the *Apriori Principle*) by “joining sibling *PrefixNodes* together”.

E.g. considers examining a new node $\langle 4, 3, 1 \rangle$ *if and only if*

- ▶ $\langle 4, 3 \rangle$ is frequent AND
 - ▶ $\langle 4, 1 \rangle$ is frequent
- Note that these are siblings.
(It does not consider $\langle 3, 1 \rangle$ – this is possible, but at a cost. see [1]).



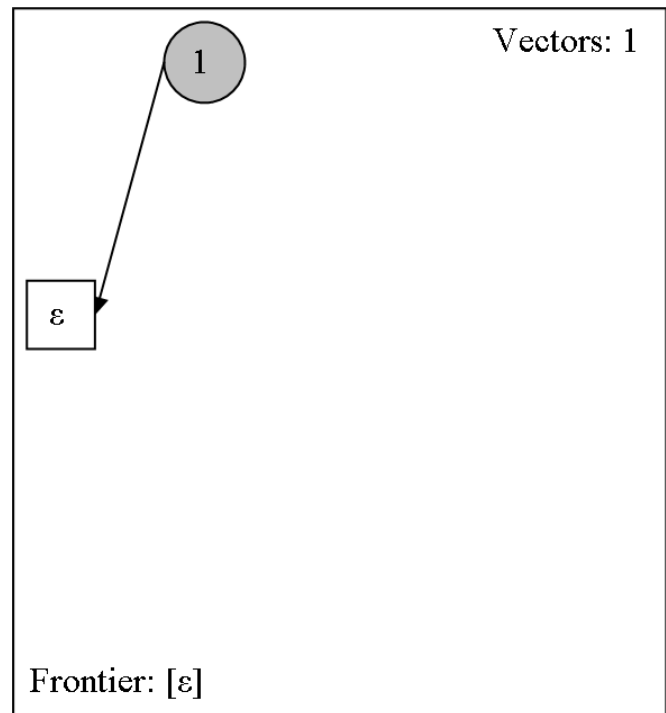
GLIMIT

- ▶ Calculates new *itemset-vectors* by \cap ing (**ANDing**) **single** *item-vectors* to an existing *itemset-vector*.
Example: $x_{\{4,3,1\}} = x_{\{4,3\}} \cap x_{\{1\}}$
rather than $x_{\{4,3,1\}} = x_{\{4\}} \cap x_{\{3\}} \cap x_{\{1\}}$ (using only single *item-vectors*) or $x_{\{4,3,1\}} = x_{\{4,3\}} \cap x_{\{4,1\}}$, etc...
- ▶ Never recalculates an *itemset-vector*
 \implies time is *linear* in the number of itemsets examined
(examining an itemset is done in constant time with respect to the number of items or size of the itemset)
- ▶ Maintains a minimum amount of *itemset-vectors* in memory.
 \implies uses space *linear* in the size of the data-set ($O(|I|)$)
(In practice – *considerably less* than the size of the data-set).
- ▶ **GLIMIT** = “**G**eometrically inspired **L**inear **I**temset **M**ining In the **T**ranspose”

Example

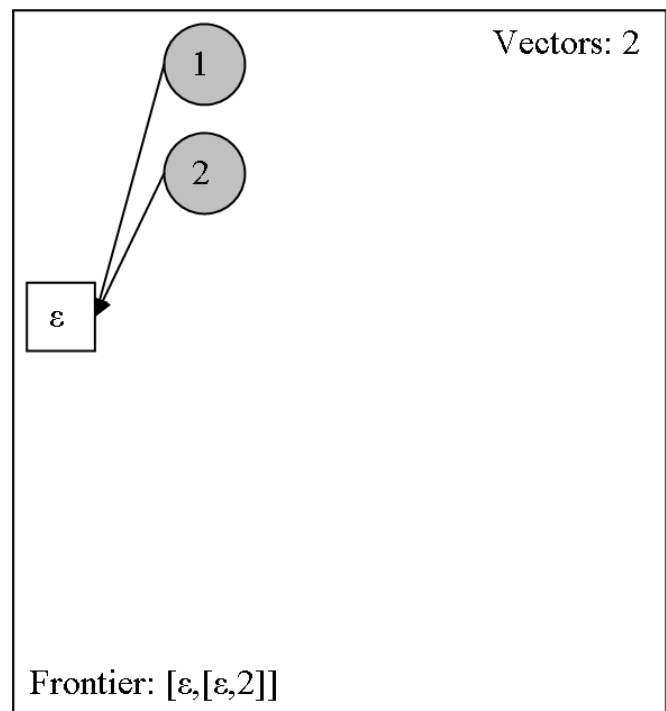
$x_{\{1\}}$

- ▶ This example assumes all itemsets are frequent.
- ▶ 1 step per slide.
- ▶ **Grey node:** the corresponding itemset-vector is in memory at the end of the step **OR** is *created* during the step (may be deleted by end of step).
- ▶ “Vectors” is the number of itemset-vectors in memory at the end of the step.
- ▶ Ignore “Frontier” (see [1]).



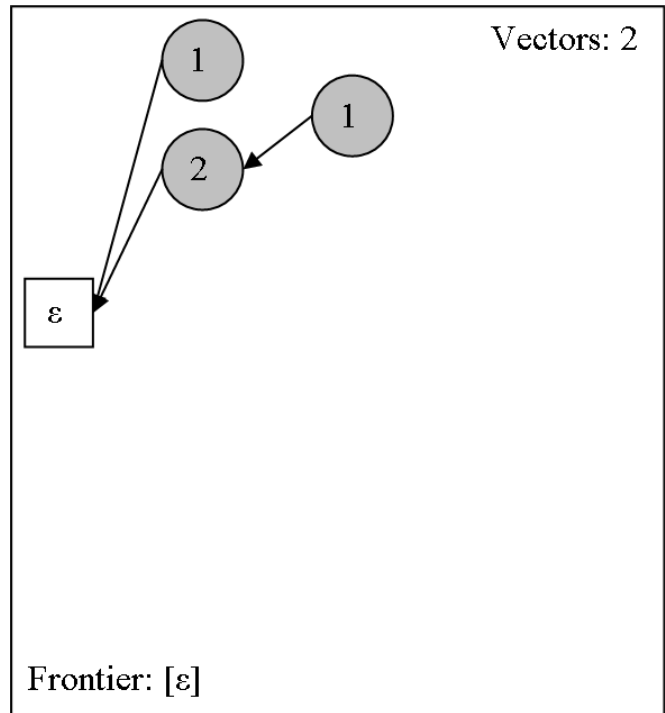
Example

$x_{\{2\}}$



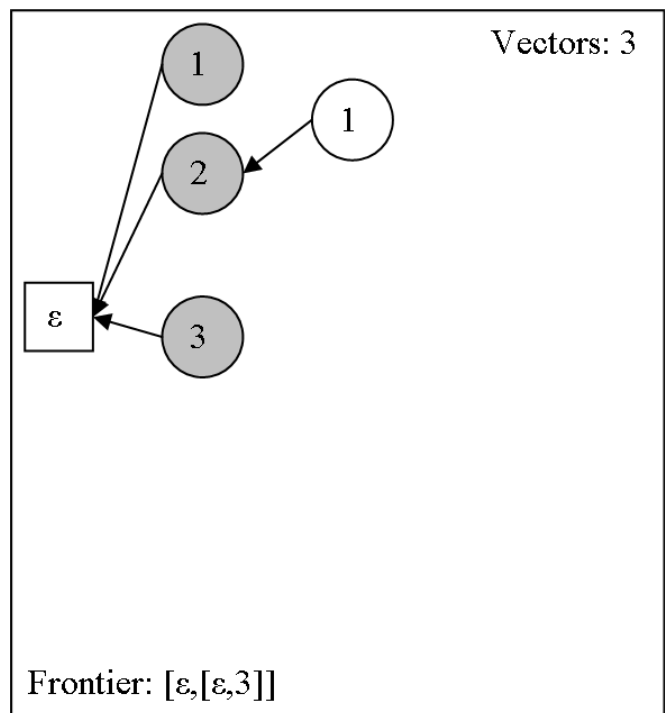
Example

$$x_{\{2,1\}} = x_{\{2\}} \cap x_{\{1\}}$$



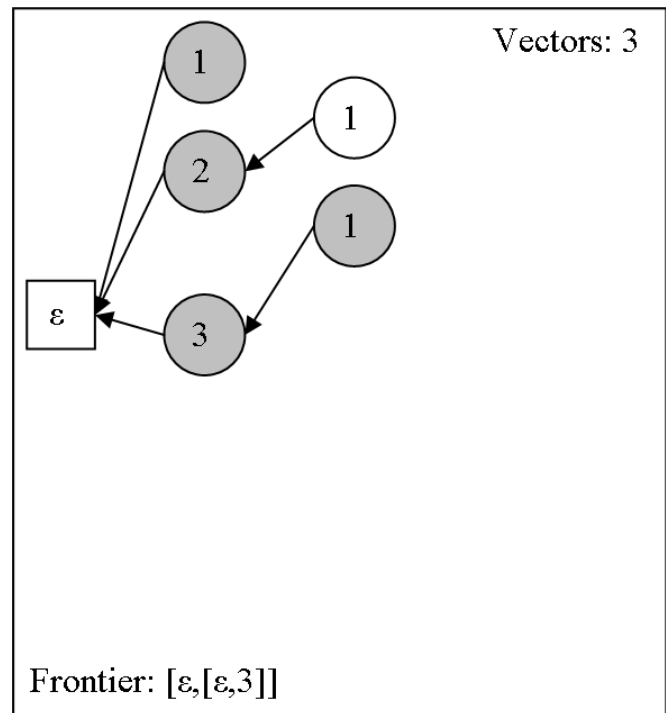
Example

$$x_{\{3\}}$$



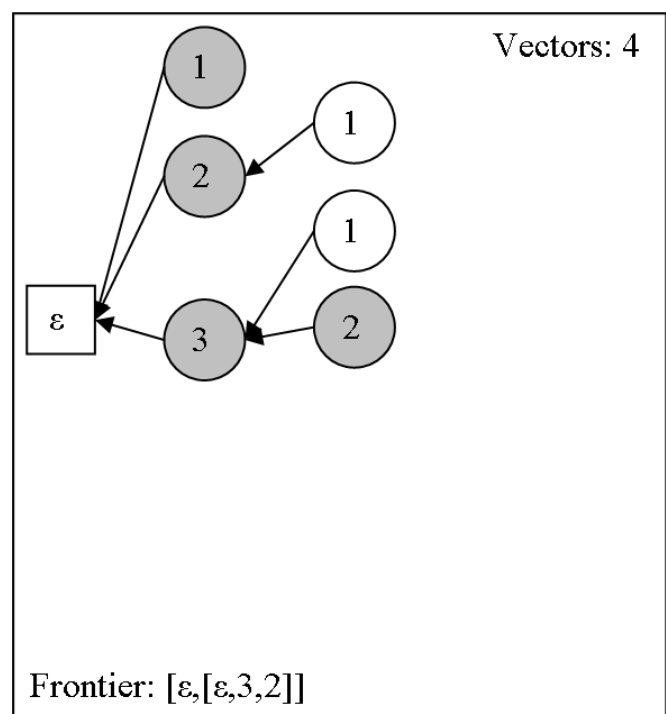
Example

$$x_{\{3,1\}} = x_{\{3\}} \cap x_{\{1\}}$$



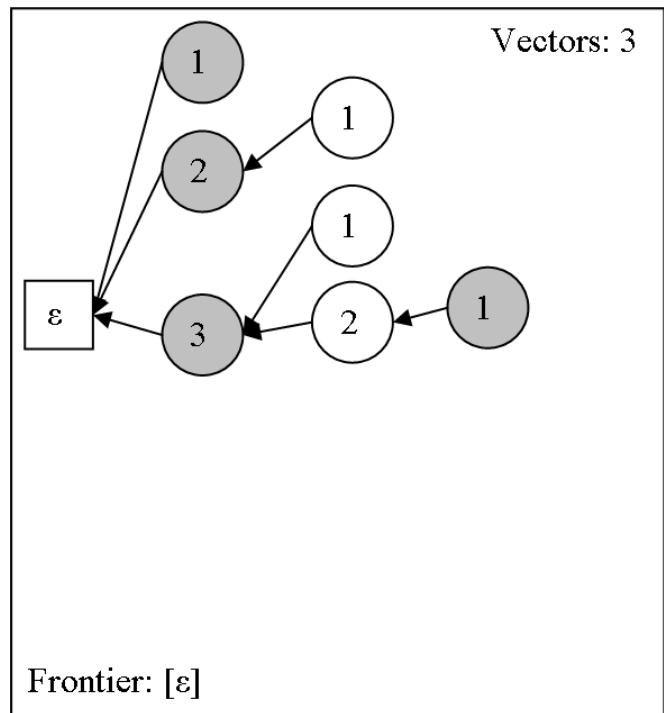
Example

$$x_{\{3,2\}} = x_{\{3\}} \cap x_{\{2\}}$$



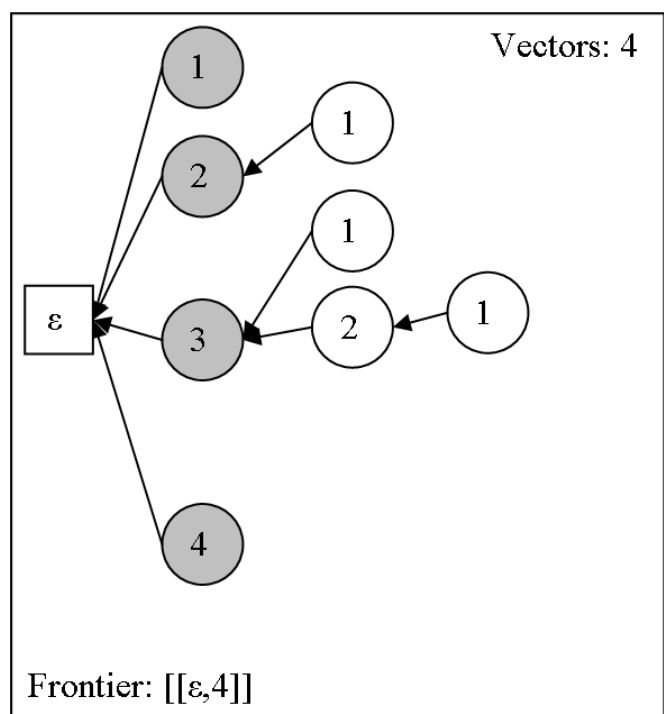
Example

$x_{\{3,2,1\}} = x_{\{3,2\}} \cap x_{\{1\}}$
 $x_{\{3,2\}}$ no longer needed



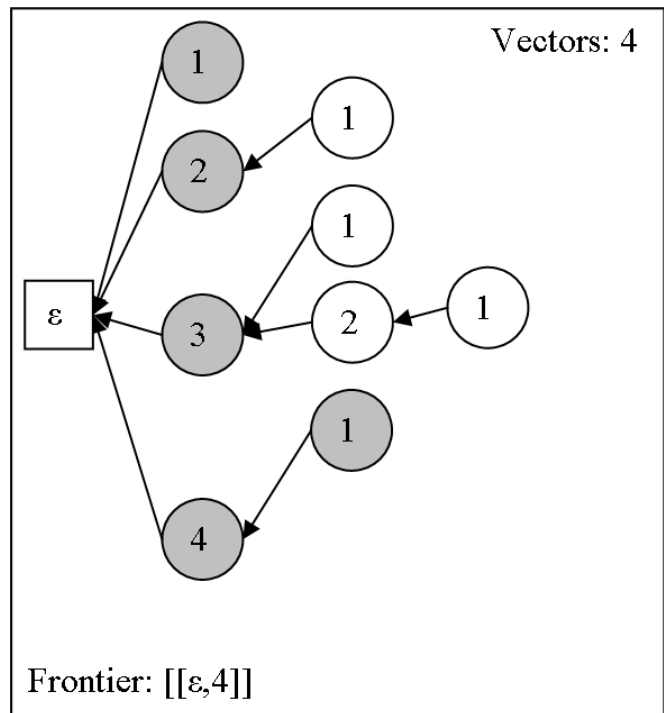
Example

$x_{\{4\}}$



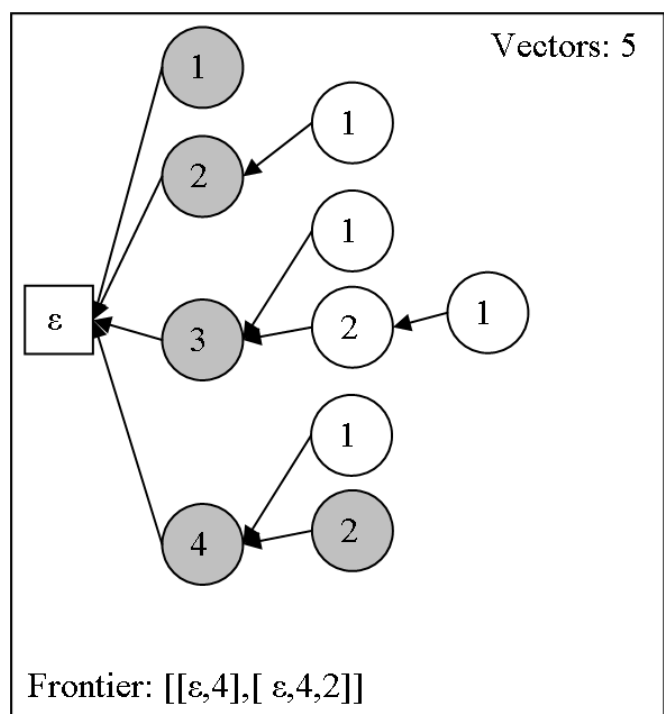
Example

$$x_{\{4,2\}} = x_{\{4\}} \cap x_{\{1\}}$$



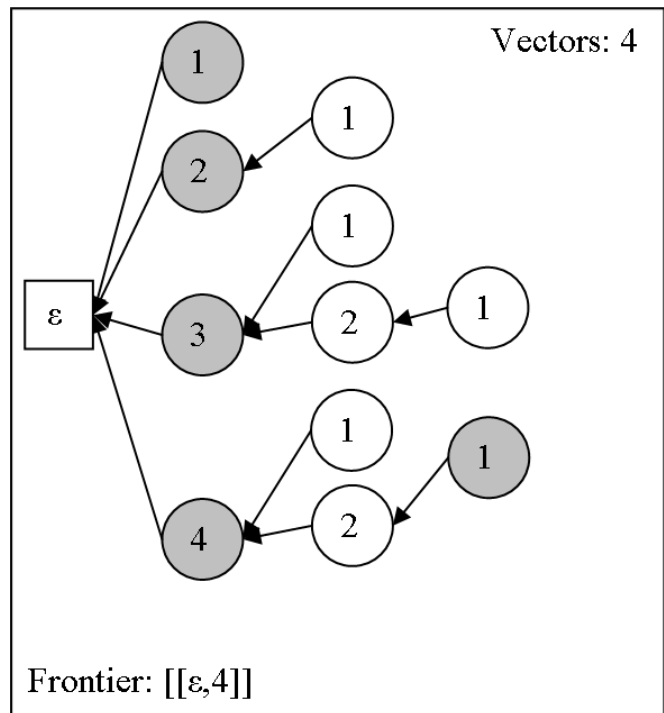
Example

$$x_{\{4,2\}} = x_{\{4\}} \cap x_{\{2\}}$$



Example

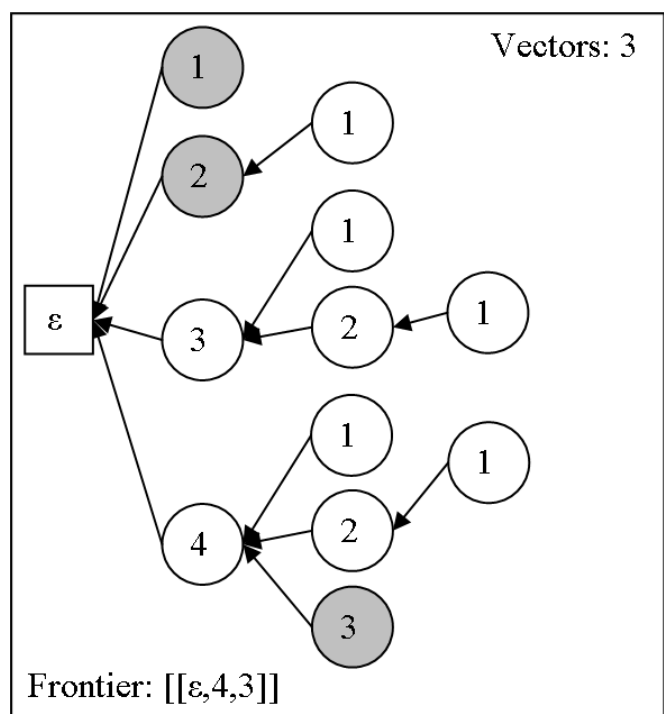
$x_{\{4,2,1\}} = x_{\{4,2\}} \cap x_{\{1\}}$
 $x_{\{4,2\}}$ no longer needed



◀ ◻ ▶ ◀ ◻ ▶ ◀ ≡ ▶ ◀ ≡ ▶ ≡ 🔍 ↻

Example

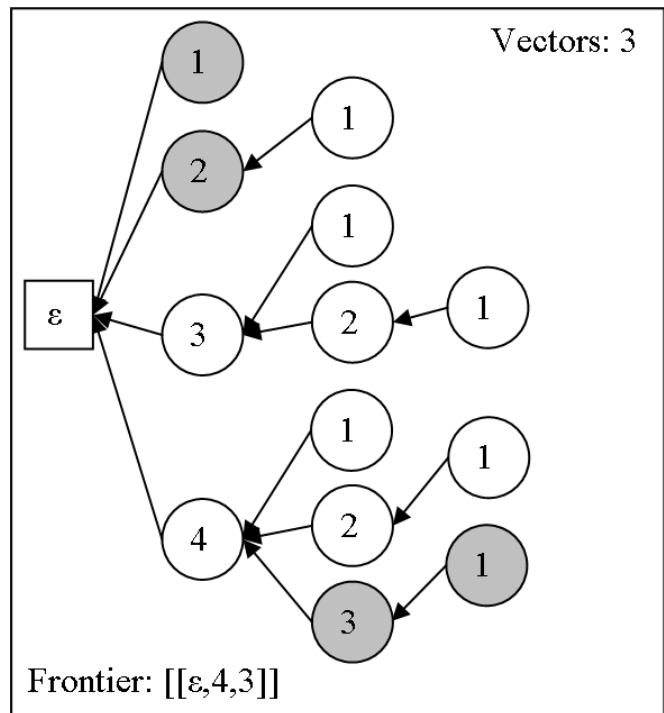
$x_{\{4,3\}} = x_{\{4\}} \cap x_{\{3\}}$
no longer need $x_{\{3\}}$ (or $x_{\{4\}}$)



◀ ◻ ▶ ◀ ◻ ▶ ◀ ≡ ▶ ◀ ≡ ▶ ≡ 🔍 ↻

Example

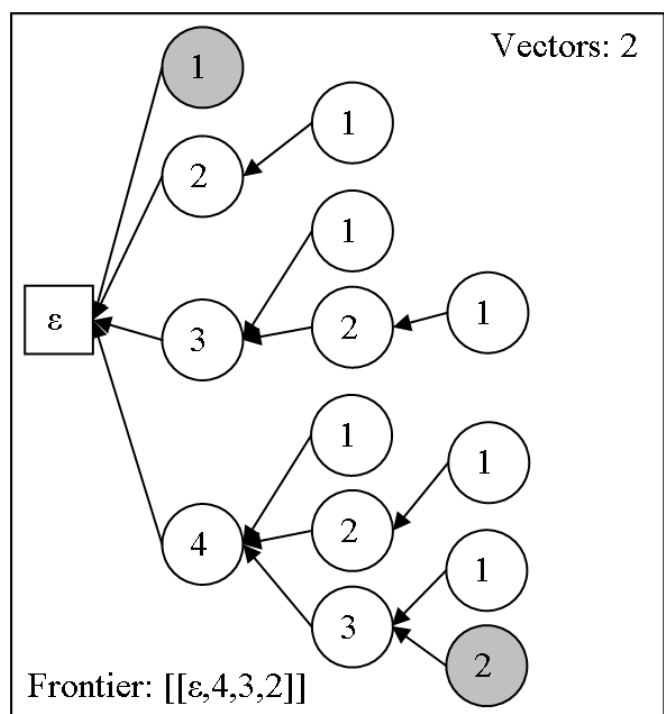
$$x_{\{4,3,1\}} = x_{\{4,3\}} \cap x_{\{1\}}$$



Example

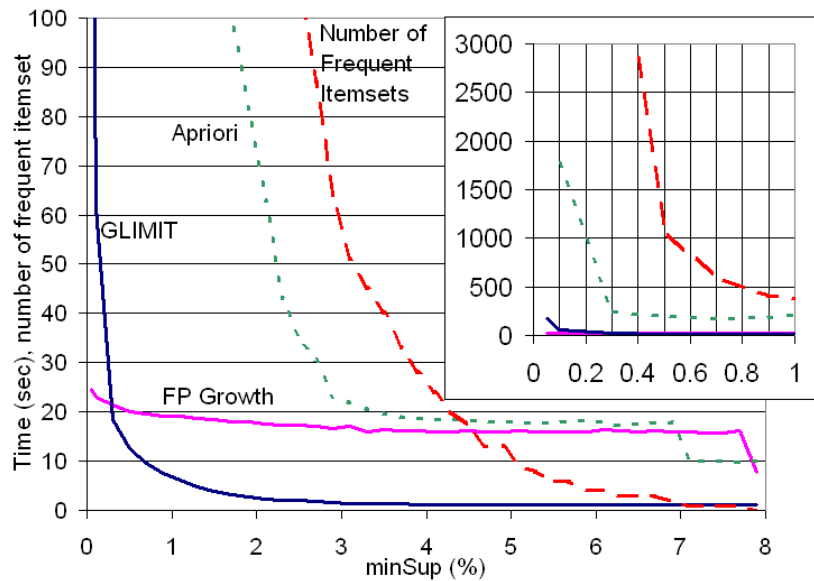
$$x_{\{4,3,2\}} = x_{\{4,3\}} \cap x_{\{2\}}$$

no longer need $x_{\{2\}}$



Run-time Performance

- ▶ Compared to FP-Growth and Apriori on data-sets from the FIMI repository (100,000 transactions, \approx 950 items).
- ▶ **Below:** Results on T10I4D100K (small) data-set. Crossover point with FP-Growth: $minSup = 0.29\%$

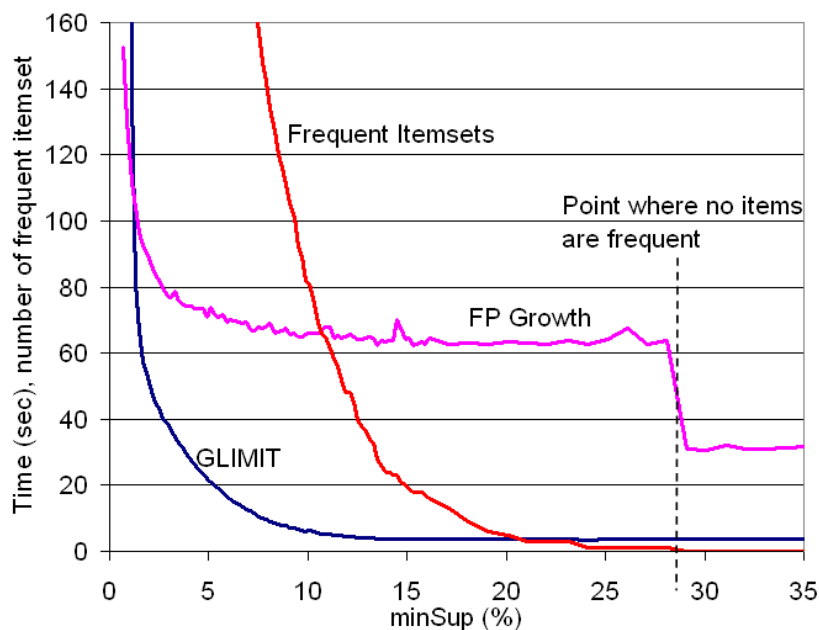


- ▶ Time to transpose data-set not included – pre-processing step independent of $minSup$. Also, note: it can actually be built into the algorithm for *no* additional time, but a little more space (but still linear space).



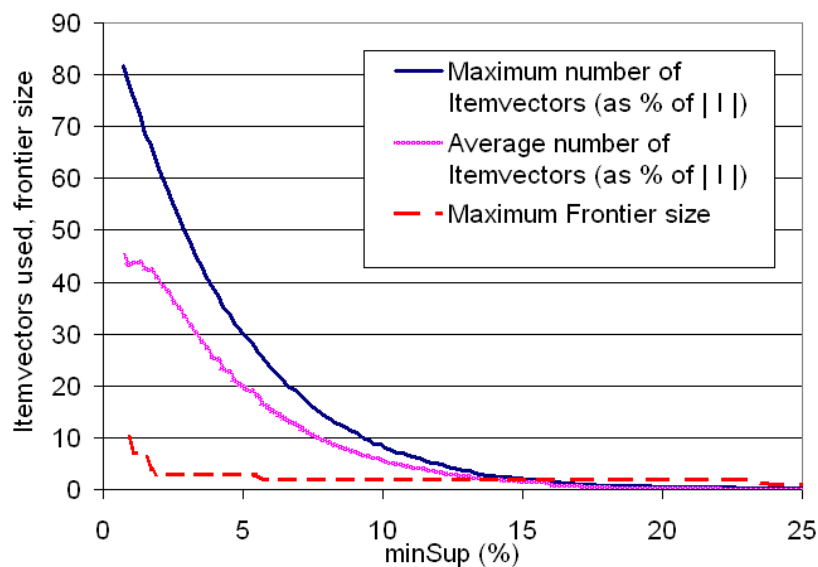
Run-time Performance

- ▶ Results on T40I10D100K (large) data-set. Crossover point with FP-Growth: $minSup = 1.2\%$.



Space Performance

- ▶ Note that space usage is much less than the size of the data-set in practice.



Ignore "Maximum frontier size".



References

- ▶ [1] Florian Verhein, Sanjay Chawla: *Geometrically Inspired Itemset Mining*. International Conference on Data Mining 2006 (ICDM'06), pp. 655-666, IEEE Computer Society, 2006.
 - ▶ Available from <http://www.it.usyd.edu.au/~fverhein/publications/>
- ▶ This is part of the work I have done for my PhD.
- ▶ Additional Questions? fverhein@it.usyd.edu.au

