Memory Efficient Itemset Tree for Targeted Association Rule Mining

Philippe Fournier-Viger[1], Espérance Mwamikazi[1],

Ted Gueniche[1] and Usef Faghihi[2]

[1]University of Moncton, Canada

[2]Sull Ross State University, USA

UNIVERSITÉ DE MONCTON

SUL ROSS STATE UNIVERSITY

15/12/2013

ADMA2013

Introduction

- Association rule mining
  - A fundamental data mining task
  - To discover **associations** in a **transaction database**
- Example:

minsup = 0.2 minconf = 0.5

Transaction database

Transaction	Items
t_1	{a, b, c, d, e}
t_2	{b, e}
t_3	{a, b, c}
t_4	{a, b, d}
t_5	{a, b, d, e}

Association rules

Association rule	Support	confidence
{a} → {d}	0.6	0.75
{a} → {b}	0.8	1.0
{a} → {c}	0.2	0.5
{b, d} → {e}	0.4	0.66
…	…	…

Algorithms

- **Popular algorithms :**
  - **Apriori, FPGrowth, Eclat, Hmine…**
- **Problems:**
  - They are *batch algorithms*: if a new transaction is added, the algorithms need to be run again.
  - Incremental algorithms solve this problem by updating **all** association rules.
  - However, not **all** association rules are always needed.
  - Users often need only a subset of all association rules at a given moment.

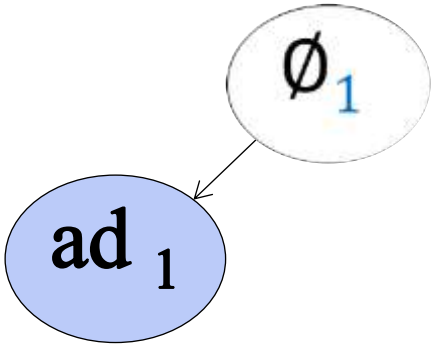
The Itemset-Tree

- **A data structure (Kubat, 2003)**
  - that can answer targeted queries for associations efficiently,
  - that can be updated efficiently.
- **Example queries**:
  - calculating the frequency of a set of items X
  - mining all association rules of the form X → ?
  - finding all frequent itemsets subsuming a given set of items X, and their support.

Construction of an Itemset Tree

Insertion of the transaction {a, d}:

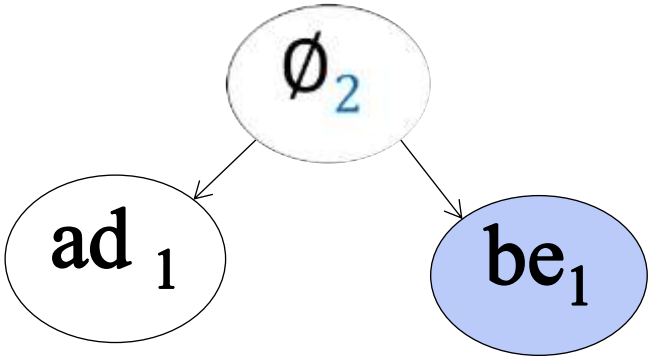
Inserted transactions	
t1	{a,d}



Construction of an Itemset Tree

Insertion of the transaction {b,e}:

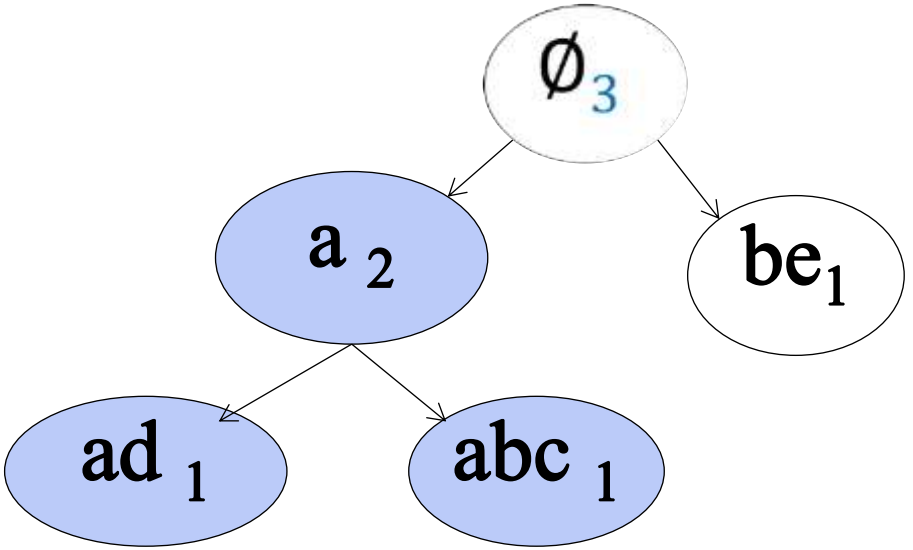
Inserted transactions	
t1	{a,d}
t2	{b, e}



Construction of an Itemset Tree

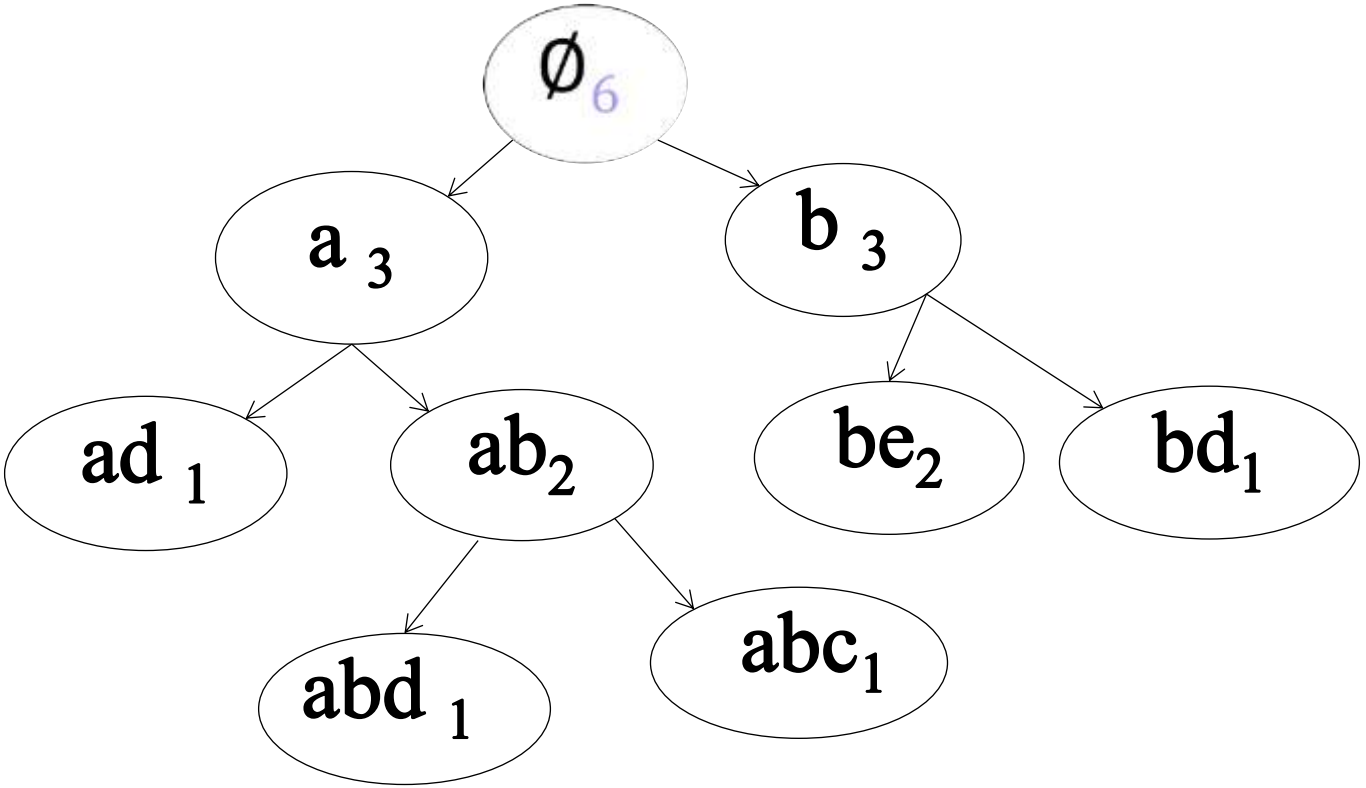
After inserting a few transactions:

Inserted transactions	
t1	{a,d}
t2	{b, e}
t3	{a, b, c}



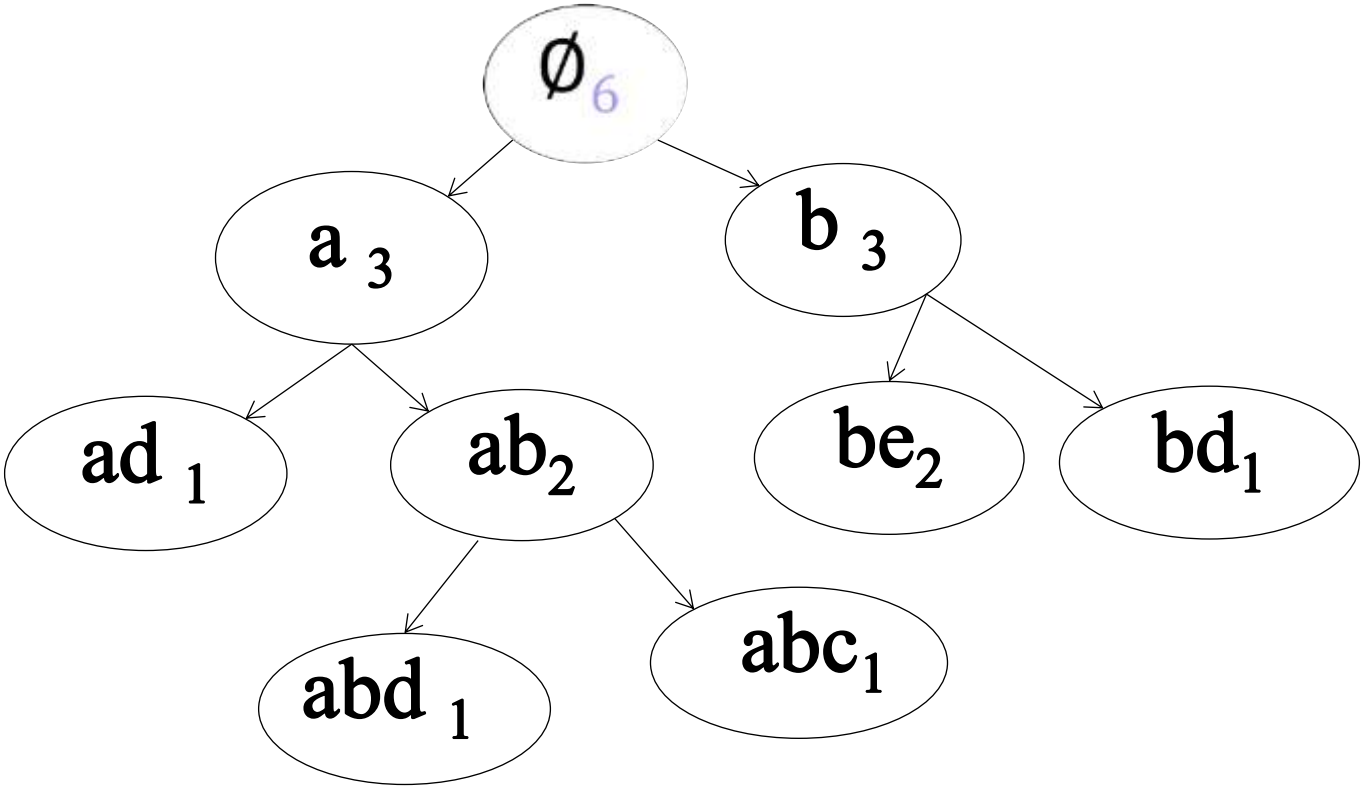
Construction of an Itemset Tree

Inserted transactions	
t1	{a,d}
t2	{b, e}
t3	{a, b, c}
t4	{a, b, d}
t5	{b, e}
t6	{b, d}



Construction of an Itemset Tree

Inserted transactions	
t1	{a,d}
t2	{b, e}
t3	{a, b, c}
t4	{a, b, d}
t5	{b, e}
t6	{b, d}

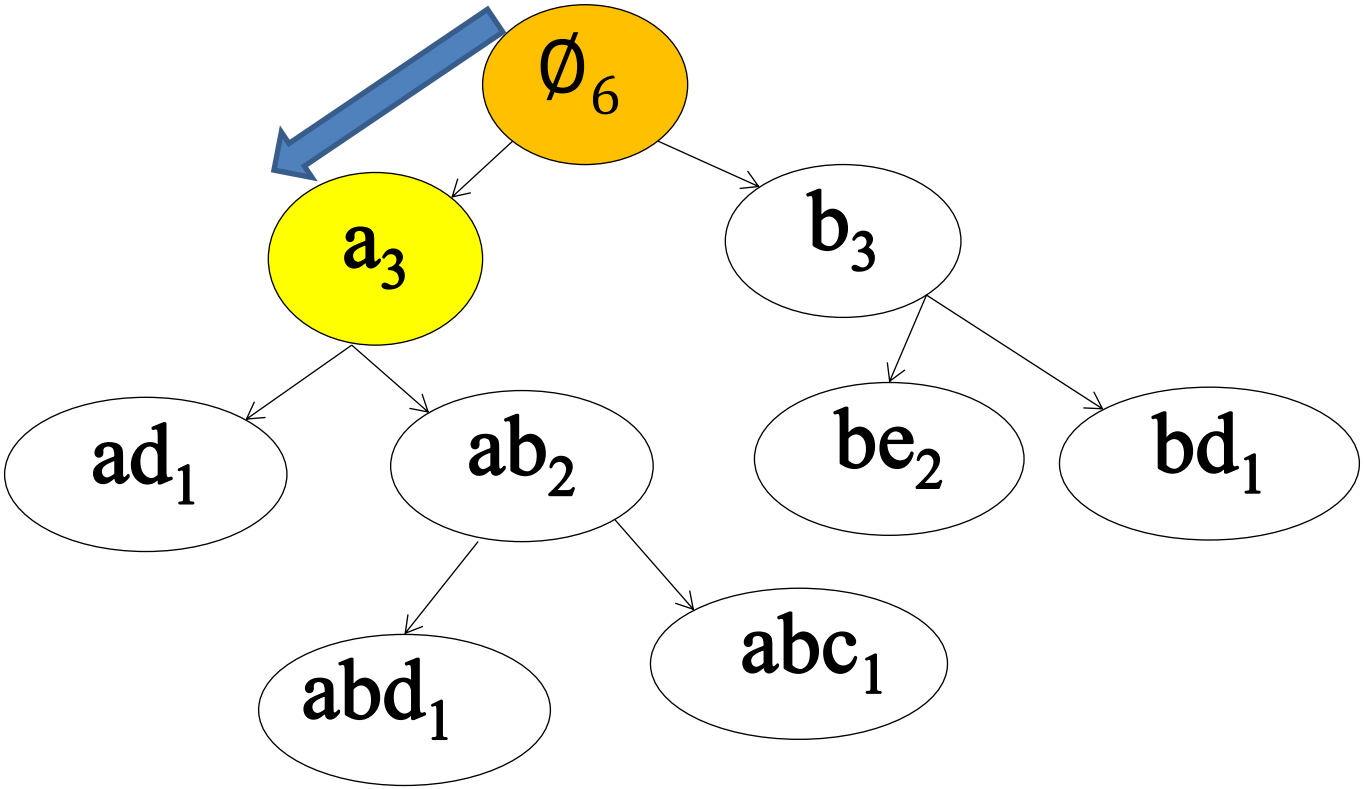


Observation 1: tree is always traversed from top to bottom for transaction insertion

Query 1: calculate the support of X={a}

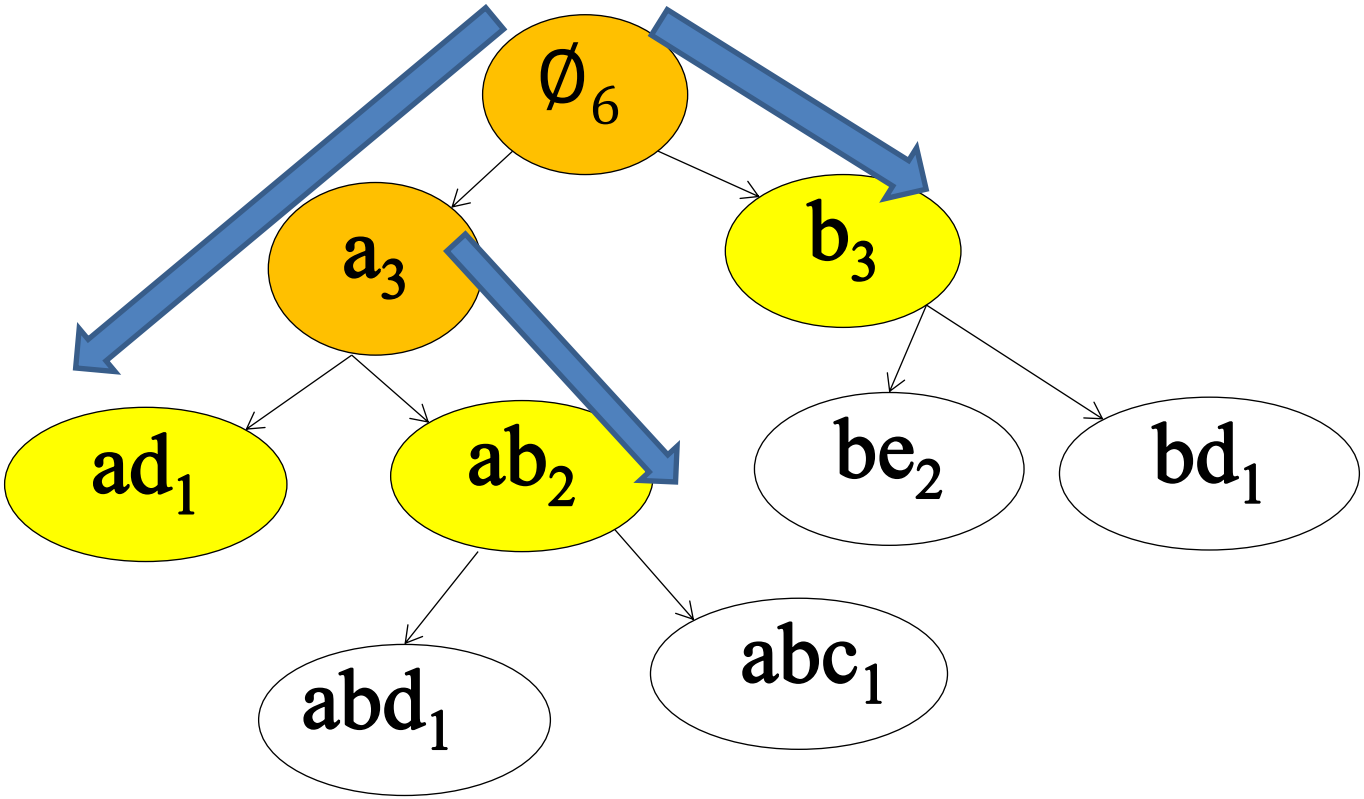
- Cost ≈ O(*n*) where *n* is |X|

Answer = 3



Query 2: calculate the support of X={a, b}

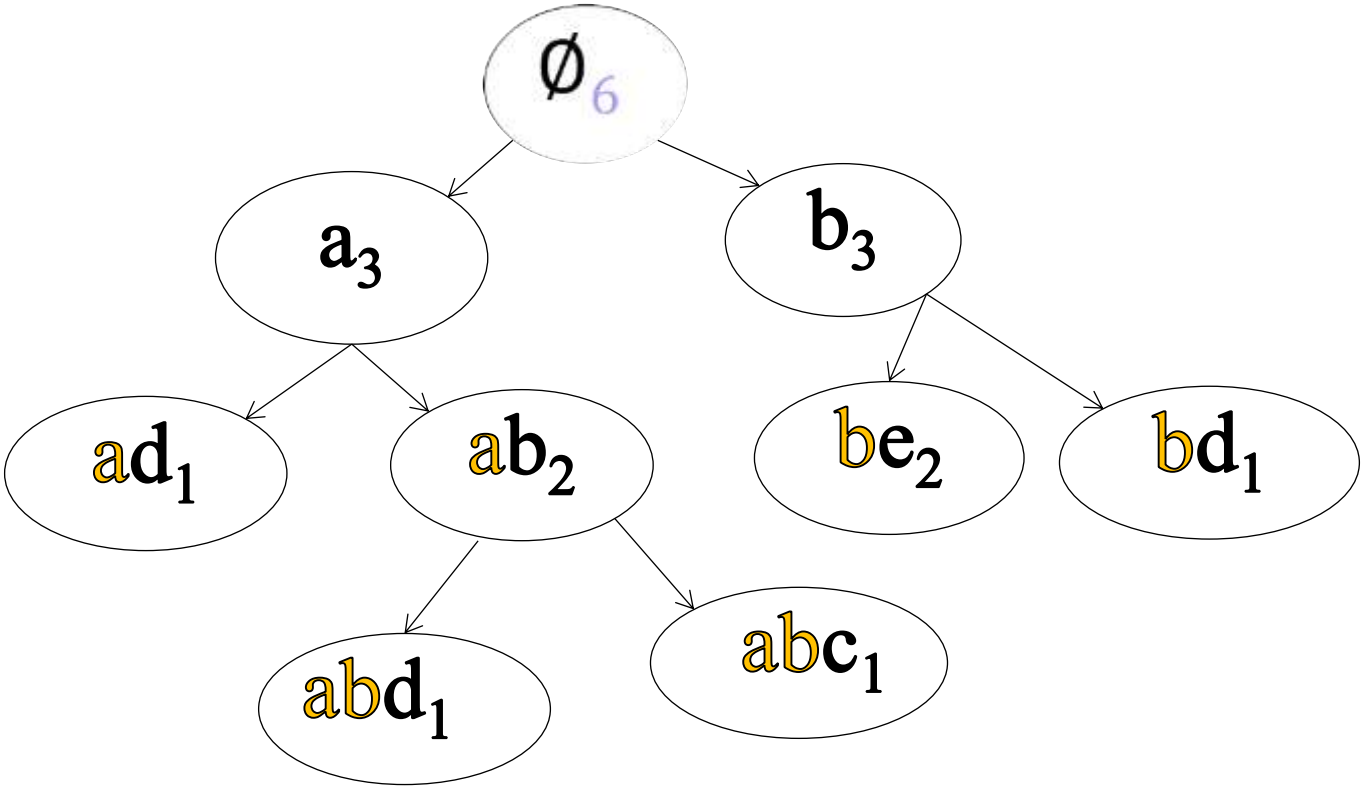
Answer = 2



Observation 2: tree is always traversed from top to bottom to answer queries

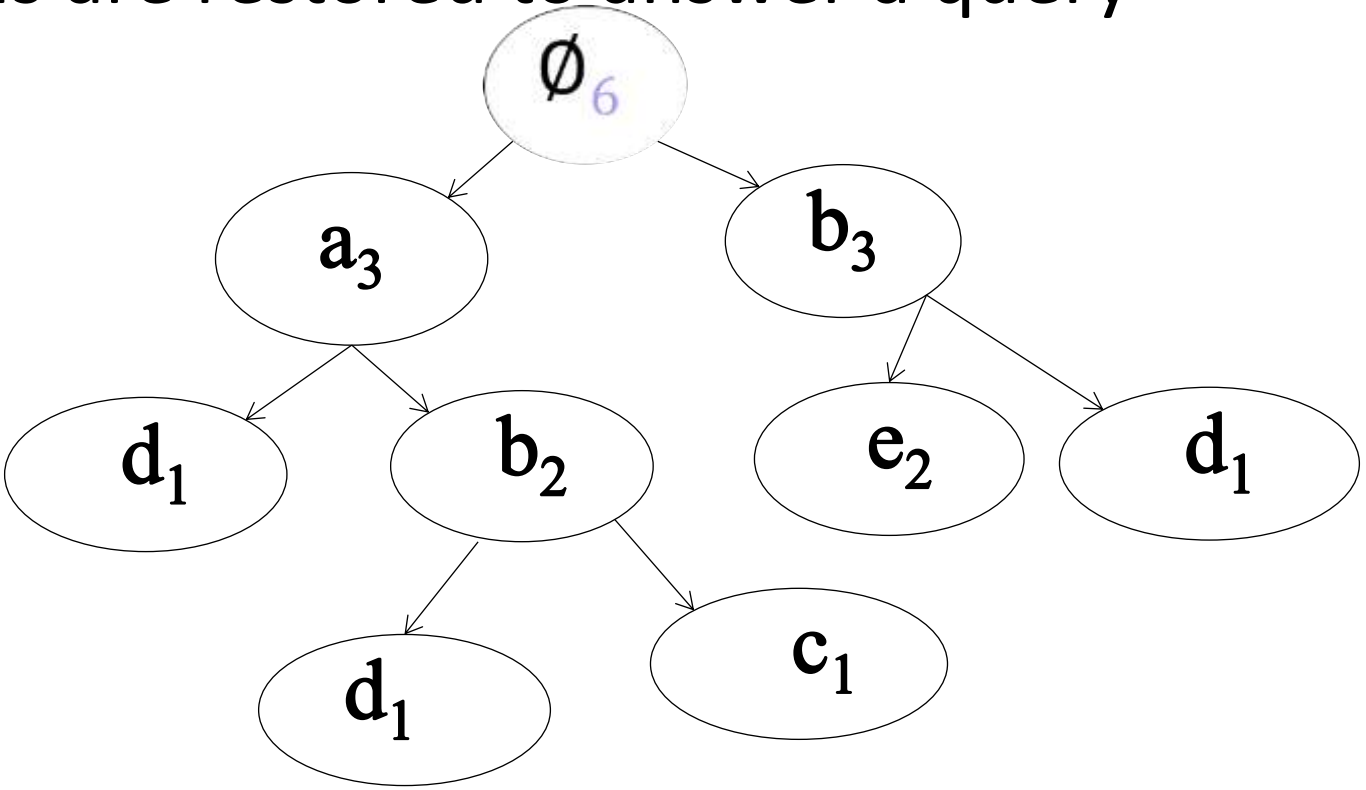
Redundancy

Observation 3 (redundancy): items appearing in a node are duplicated in all descendent nodes



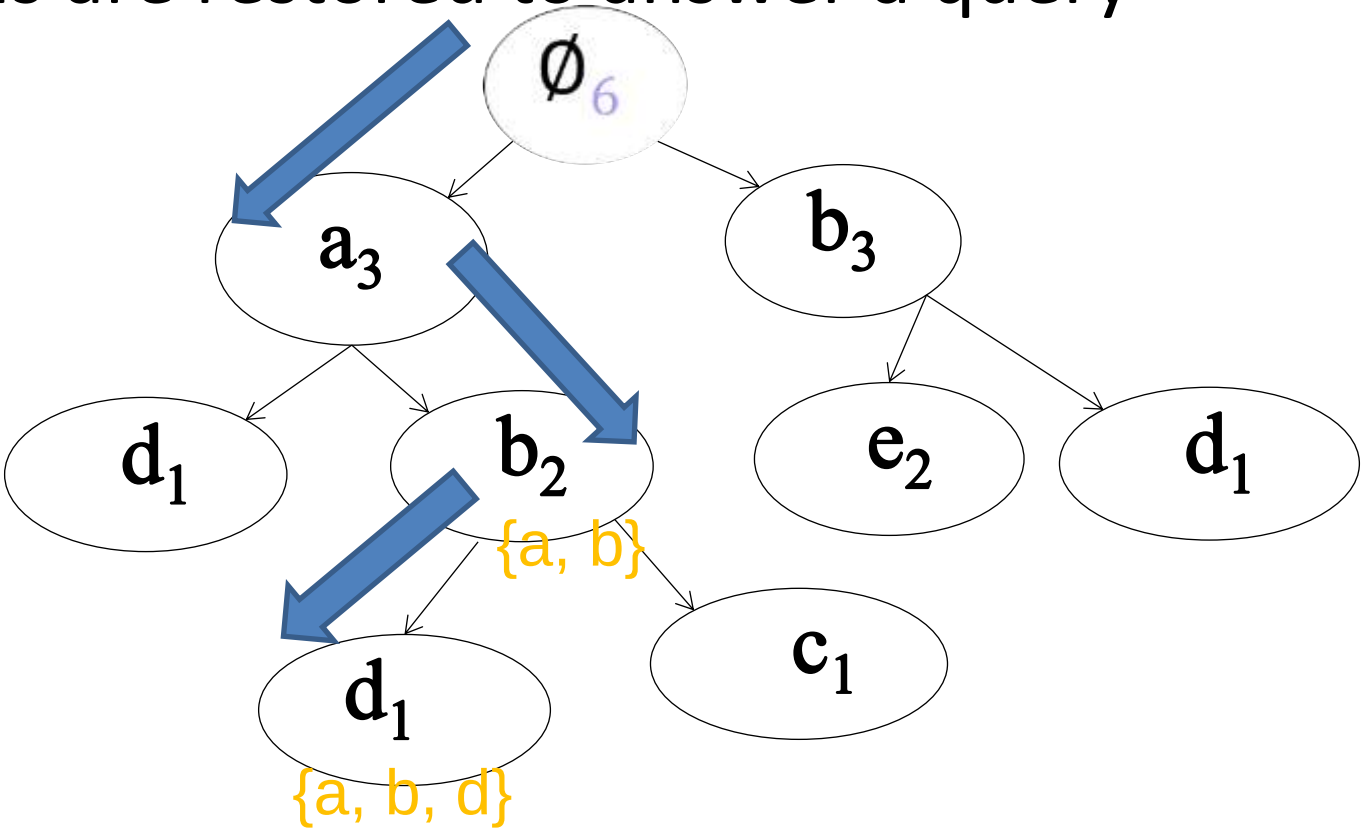
Our proposal : MEIT (Memory Efficient Itemset Tree)

- Redundant items are removed during construction
- Redundant items are restored to answer a query



Our proposal : MEIT (Memory Efficient Itemset Tree)

- Redundant items are removed during construction
- Redundant items are restored to answer a query



Experiment 1 – Memory Usage

Datasets

Dataset	Transaction count	Distinct items count	Average transaction size
Accidents	340,183	468	22
C73D10K	10,000	1,592	73
Chess	3,196	75	37
Connect	67,557	129	43
Mushrooms	8,416	128	23
Pumsb	49,046	7,116	74
Retail	88,162	16,470	172

Memory Usage (total node size)

Dataset	IT size (items)	MEIT size (items)	Size reduction (%)
Accidents	16057697	5262555	67.2%
C73D10K	882196	508878	42.3%
Chess	196579	39550	79.9%
Connect	4446791	1092208	75.4%
Mushrooms	308976	35003	88.7%
Pumsb	4046316	2773440	31.5%
Retail	938568	690889	26.4%

Average reduction: 58. 3 %

Experiment 1 – Memory Usage

Memory Usage (MB)

Dataset	IT size (MB)	MEIT size (MB)	Size reduction (%)
Accidents	84.1	43.0	49%
C73D10K	4.0	2.6	36%
Chess	1.0	0.4	60%
Connect	21.8	9.0	59%
Mushrooms	1.7	0.7	60%
Pumsb	18.3	13.5	26%
Retail	7.3	6.4	13%

Average reduction: 43 %

Experiment 2 – processing overhead

Dataset	Tree construction time (s)		Time for processing 10K queries (s)	
	IT	MEIT	IT	MEIT
Accidents	3.71	5.82	880.4	1696.8
C73D10K	0.25	0.38	23.8	39.9
Chess	0.22	0.30	2.6	5.0
Connect	0.56	0.82	153.5	231.1
Mushrooms	0.21	0.23	1.0	2.1
Pumsb	0.63	0.95	134.2	202.1
Retail	4.33	7.75	84.8	193.7

- transaction insertion overhead: ≈45%
- query processing time overhead: ≈44%
- overhead is predictable and is a trade-off

Conclusion

- Memory Efficient Itemset Tree
  - on-the-fly node compresssion and decompression to reduce tree size,
  - compatible with existing querying algorithms
  - up to 43 % smaller than an Itemset Tree on average
- Source code available as part of the **SPMF data mining library (**GPL 3)
- **Future work**: compression of tree structure, caching algorithms for frequently accessed nodes…

Open source Java data mining software, 55 algorithms
http://www.phillippe-fournier-viger.com/spmf/

Thank you. Questions?

Open source Java data mining software, 55 algorithms
http://www.phillippe-fournier-viger.com/spmf/