Data Mining: Foundation, Techniques and Applications

Lesson 5,6: Association Rules/Frequent Patterns

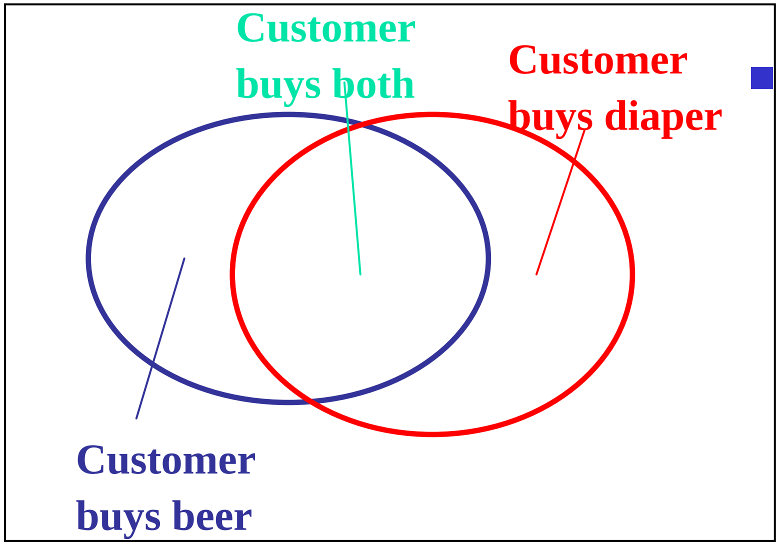
Li Cuiping(李翠平)
School of Information
Renmin University of China

Anthony Tung(鄧锦浩)
School of Computing
National University of Singapore

Association Rule: Basic Concepts

- Given: (1) database of transactions, (2) each transaction is a list of items (purchased by a customer in a visit)
- Find: all rules that correlate the presence of one set of items with that of another set of items
  - E.g., *98% of people who purchase tires and auto accessories also get automotive services done*
- Applications
  - ** ⇒ Maintenance Agreement* (What the store should do to boost Maintenance Agreement sales)
  - *Home Electronics ⇒ ** (What other products should the store stocks up?)
  - Attached mailing in direct marketing
  - Detecting "ping-pong"ing of patients, faulty "collisions"

Rule Measures: Support and Confidence



Find all the rules *X & Y* $\Rightarrow$ *Z* with minimum confidence and support

- support, *s*, probability that a transaction contains {X & Y & Z}
- confidence, *c*, conditional probability that a transaction having {X & Y} also contains *Z*

Transaction ID	Items Bought
2000	A,B,C
1000	A,C
4000	A,D
5000	B,E,F

Let minimum support 50%, and minimum confidence 50%, we have

- *A* $\Rightarrow$ *C* (50%, 66.6%)
- *C* $\Rightarrow$ *A* (50%, 100%)

Mining Association Rules—An Example

TID	Items
10	a, c, d
20	b, c, e
30	a, b, c, e
40	b, e

Frequent Itesmset	Support
a	2
b	3
c	3
e	3
ac	2
bc	2
be	3
ce	2
bce	2

Min. support 50%
Min. confidence 50%

For rule $a \Rightarrow c$

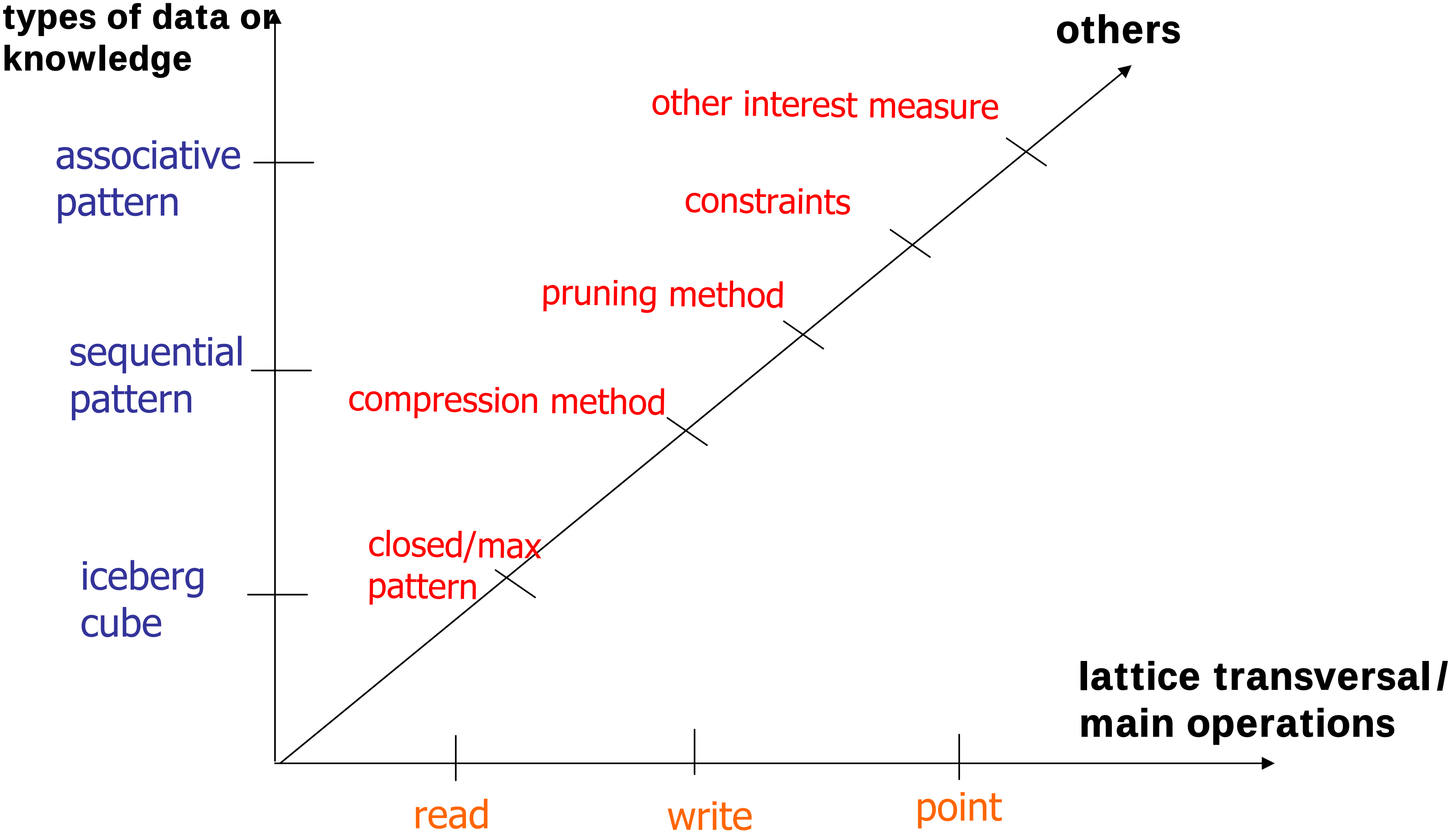
support = support({a c}) = 50%

confidence = support({a c})/support({a}) = 66.6%

Mining Frequent Itemsets: the Key Step

- Find the *frequent itemsets*: the sets of items that have minimum support
  - A subset of a frequent itemset must also be a frequent itemset
    - i.e., if {a b} is a frequent itemset, both {a} and {b} should be a frequent itemset
  - Iteratively find frequent itemsets with cardinality from 1 to *k* *(k*-itemset*)*
- Use the frequent itemsets to generate association rules.

A Multidimensional View of Frequent Patten Discovery



Data and Knowledge Types

- Associative Pattern
  - transactional table vs relational table
  - boolean vs quantitative
- Sequential Pattern
  - A *sequence* : < (ef) (ab) (df) c b >
  - (e,f)->(a,b)-> c occur 50% of the time
- Iceburg Cube
  - table with a measures

TID	Items
10	a, c, d
20	b, c, e
30	a, b, c, e
40	b, e

→

TID	a	b	c	d	e
10	1	0	1	1	0
20	0	0	1	0	1
30	1	1	1	0	1
40	0	1	0	0	1

transactional=binary

Month	City	Cust_grp	Prod	Cost	Price
Jan	Tor	Edu	Printer	500	485
Mar	Van	Edu	HD	540	520
…	…	…	…	…	…

relational with quantitative attribute

Month	City	Cust_grp	Prod	Cost **(Support)**
Jan	Tor	*	Printer	1040
…	…	…	…	…

cube: using other measure as support

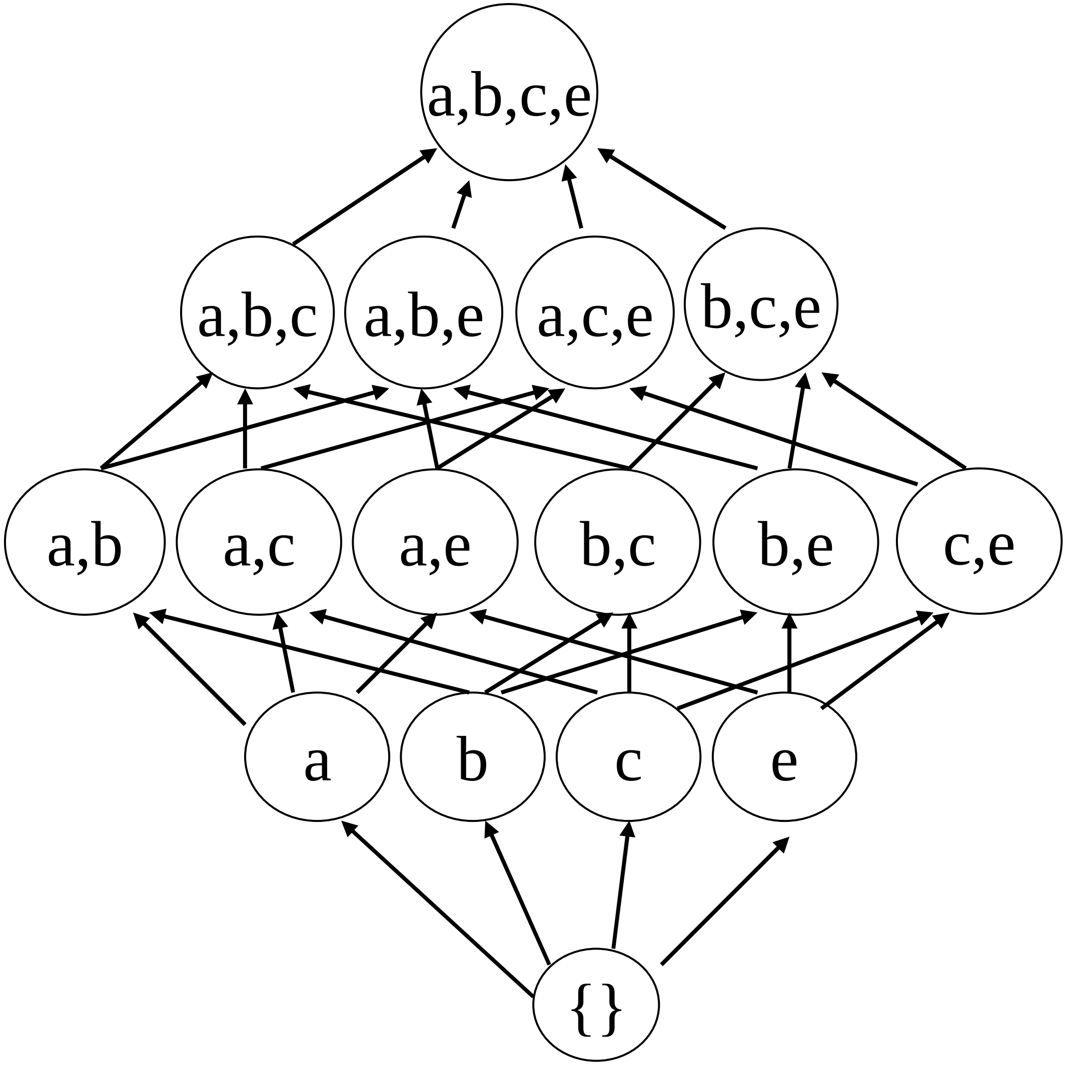
Simulation of Lattice Transversal

The whole process of frequent pattern mining can be seen as a search in the lattice. Variety come in

- Breadth first vs Depth first

- Bottom up vs top down

- Read-based, Write-Based, Pointer-Based



Three categories of FPM Algorithms.

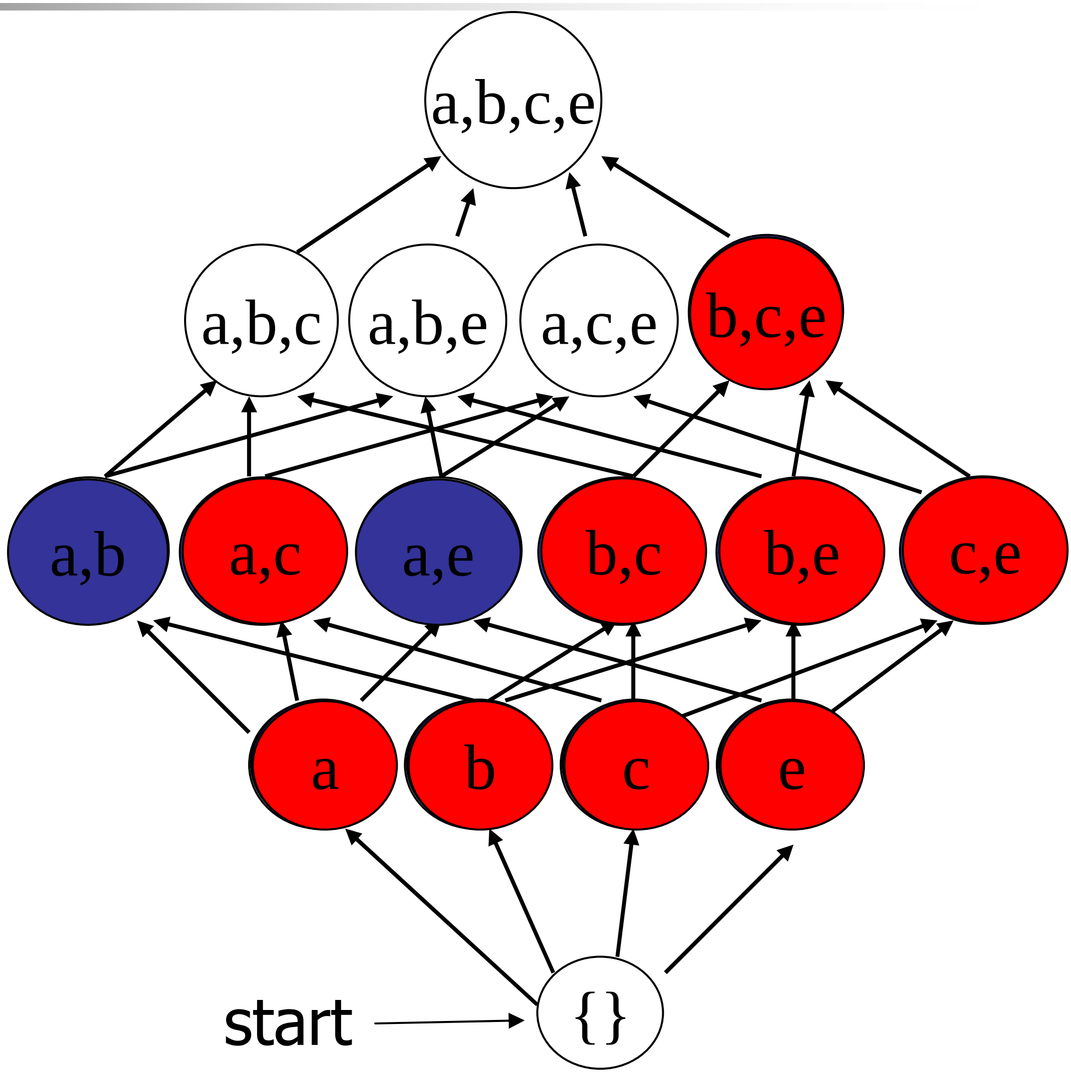
- Read [AgSr94]
  - Apriori-based. No write involved.
- Write [Zaki00, HPY00, HaPe00]
  - Perform write to improve performance.
- Point [BeRa99, HPDW01]
  - Memory based algorithm designed to point to various part of the data instead of re-writing the data in another part of the memory.

Generalized Framework

	Read-based	Write-based	Point-based
Association Mining	Apriori[AgSr94]	Eclat, MaxClique[Zaki01], FPGrowth [HaPe00]	
Sequential Pattern Discovery	GSP[AgSr96]	SPADE [Zaki98,Zaki01], PrefixSpan [PHPC01]	Hmine
Iceberg Cube	Apriori[AgSr94]		BUC[BeRa99], H-cubing [HPDW01]

The Apriori Algorithm

- Bottom-up, breadth first search
- Only read is perform on the databases
- Store candidates in memory to simulate the lattice search
- Iteratively follow the two steps:
  - generate candidates
  - count and get actual frequent items



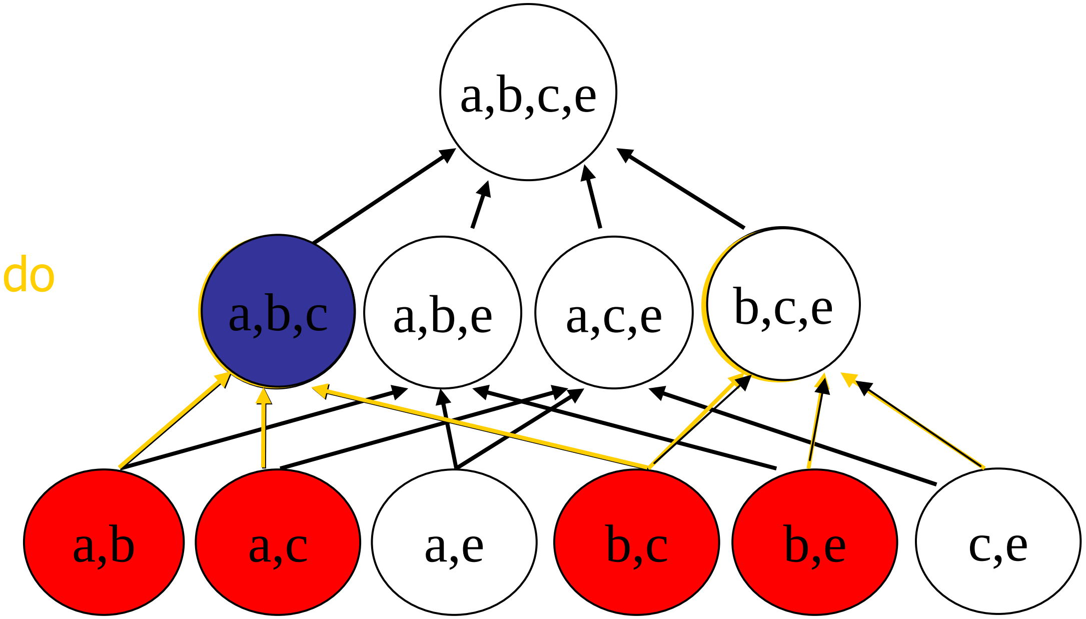
Candidate Generation and Pruning

- Suppose all frequent (k-1) items are in L_{k-1}
- Step 1: Self-joining L_{k-1}

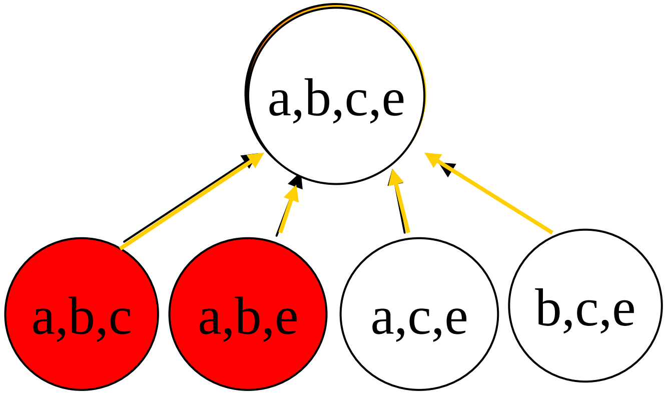
```
insert into $C_k$
select $p.i_1, p.i_2, \ldots, p.i_{k-1}, q.i_{k-1}$
from $L_{k-1}$ p, $L_{k-1}$ q
where $p.i_1=q.i_1$ , … , $p.i_{k-2}=q.i_{k-2}$ , $p.i_{k-1} < q.i_{k-1}$
```

- Step 2: pruning

```
forall itemsets c in $C_k$ do
    forall (k-1)-subsets s of c do
        if (s is not in $L_{k-1}$) then
            delete c from $C_k$
```



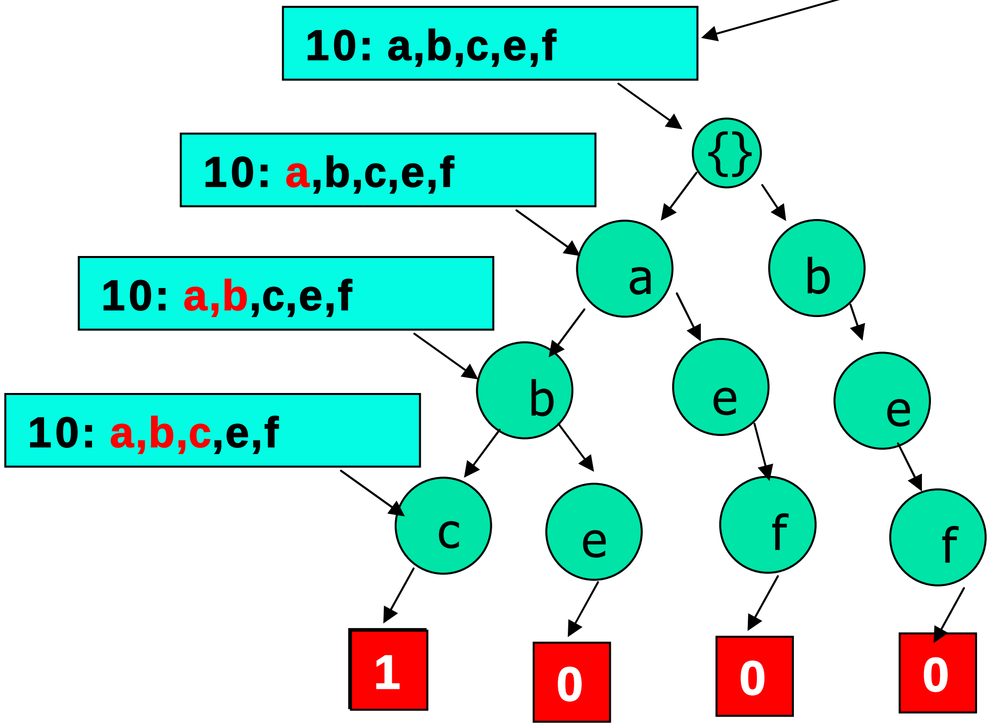
Candidate Generation and Pruning(another example)



Counting Supports

- Stored candidates itemsets in a trier structure for support counting

TID	Items
10	a, b, c, e, f
20	b, c, e
30	b, f, g
40	b, e
. .	. .



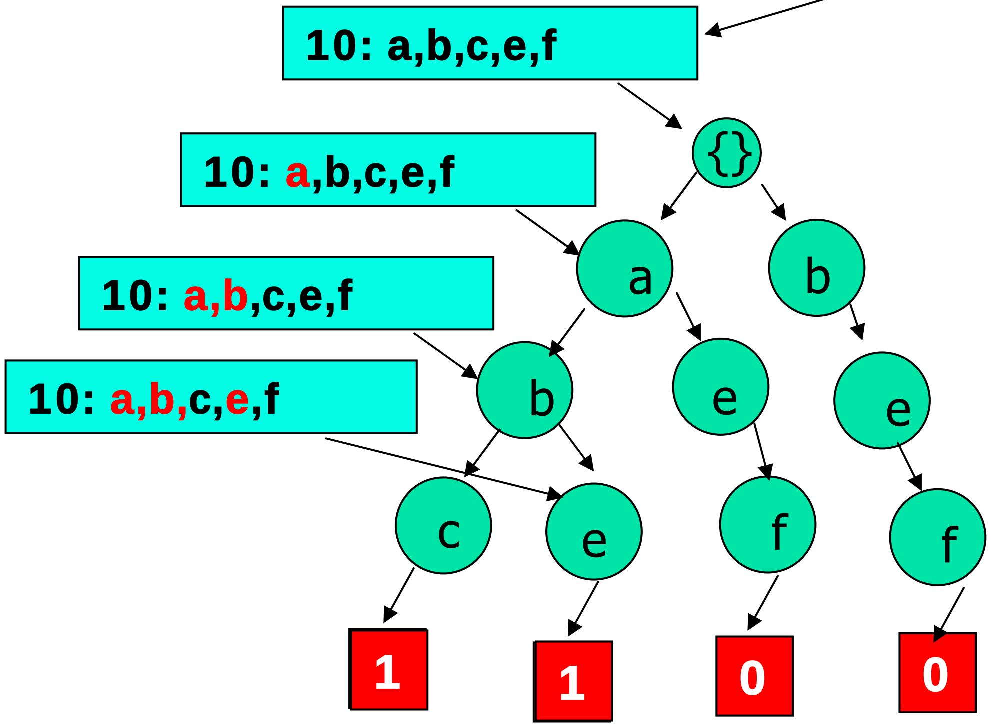
Candidate Itemsets:

{a,b,c}, {a,b,e}, {a,e,f}, {b,e,f}

Counting Supports

- Stored candidates itemsets in a trier structure for support counting

TID	Items
10	a, b, c, e, f
20	b, c, e
30	b, f, g
40	b, e
:	:



10: a,b,c,e,f

10: a,b,c,e,f

10: a,b,c,e,f

10: a,b,c,e,f

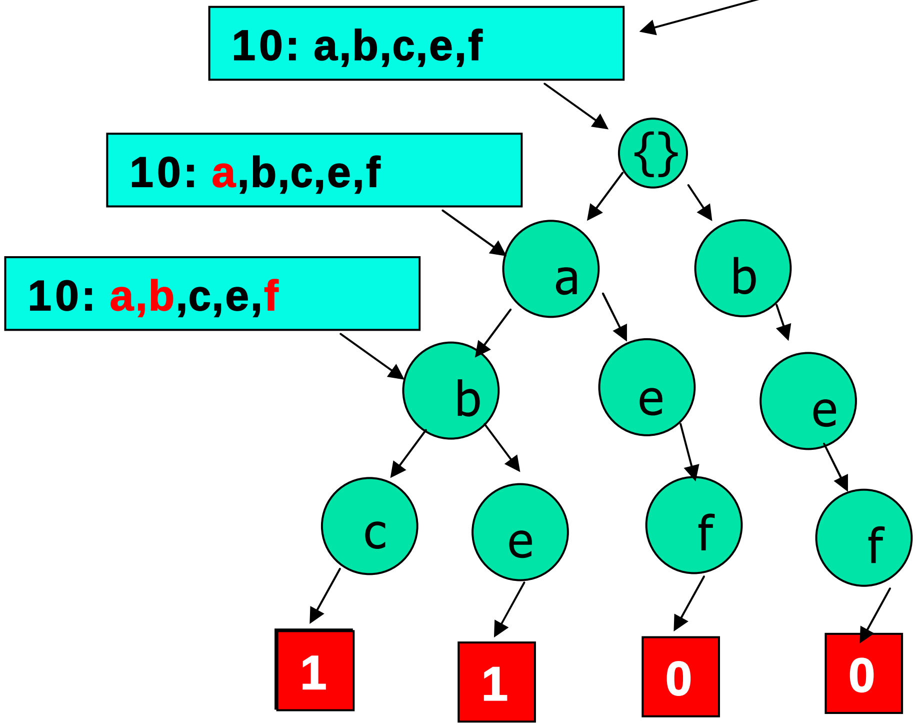
Candidate Itemsets:

{a,b,c}, {a,b,e}, {a,e,f}, {b,e,f}

Counting Supports

- Stored candidates itemsets in a trier structure for support counting

TID	Items
10	a, b, c, e, f
20	b, c, e
30	b, f, g
40	b, e
:	:



10: a,b,c,e,f

10: a,b,c,e,f

10: a,b,c,e,f

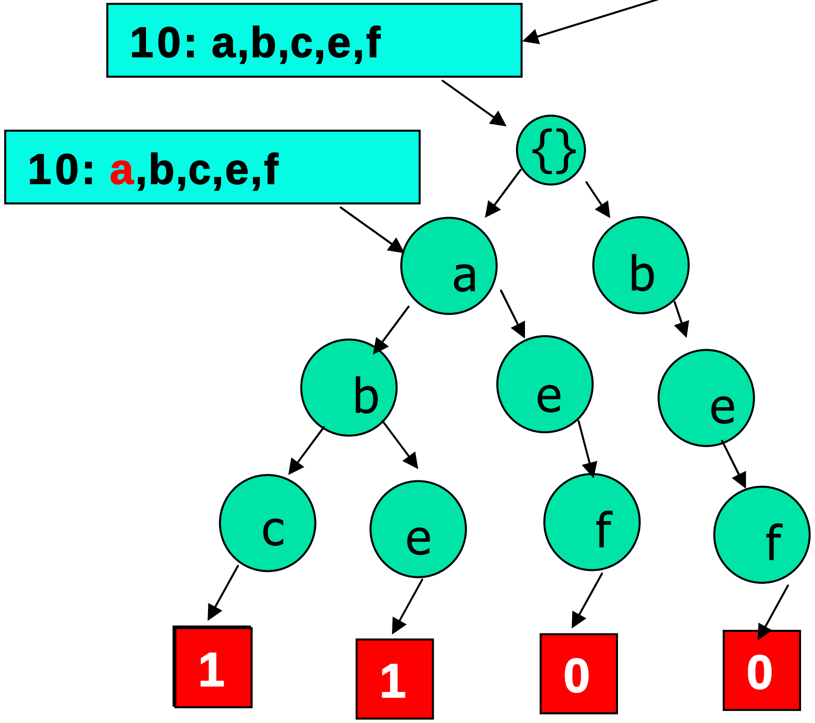
Candidate Itemsets:

{a,b,c}, {a,b,e}, {a,e,f}, {b,e,f}

Counting Supports

- Stored candidates itemsets in a trier structure for support counting

TID	Items
10	a, b, c, e, f
20	b, c, e
30	b, f, g
40	b, e
:	:



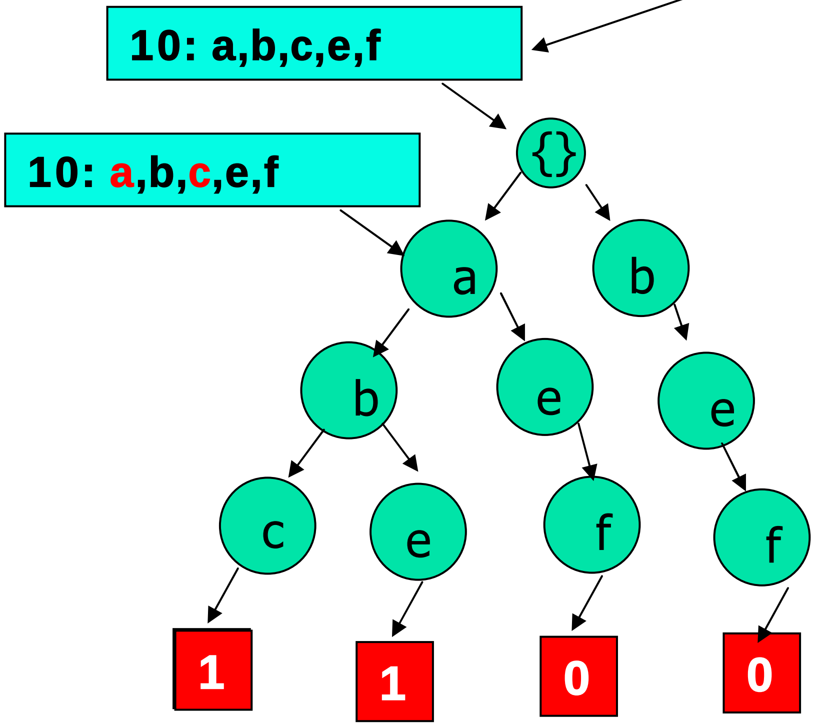
Candidate Itemsets:

{a,b,c}, {a,b,e}, {a,e,f}, {b,e,f}

Counting Supports

- Stored candidates itemsets in a trier structure for support counting

TID	Items
10	a, b, c, e, f
20	b, c, e
30	b, f, g
40	b, e
:	:



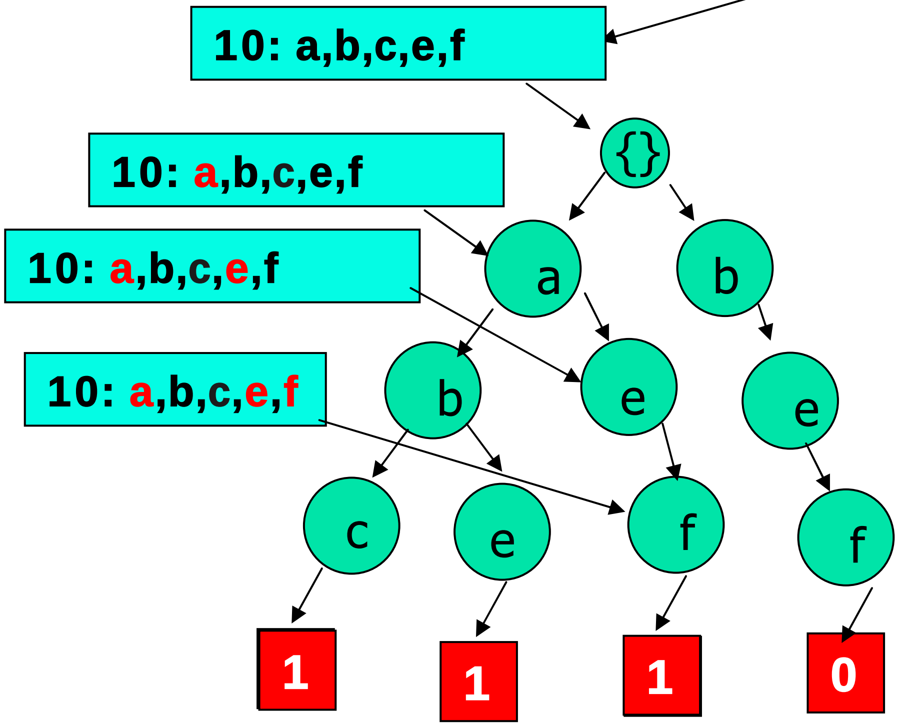
Candidate Itemsets:

{a,b,c}, {a,b,e}, {a,e,f}, {b,e,f}

Counting Supports

- Stored candidates itemsets in a trier structure for support counting

TID	Items
10	a, b, c, e, f
20	b, c, e
30	b, f, g
40	b, e
:	:

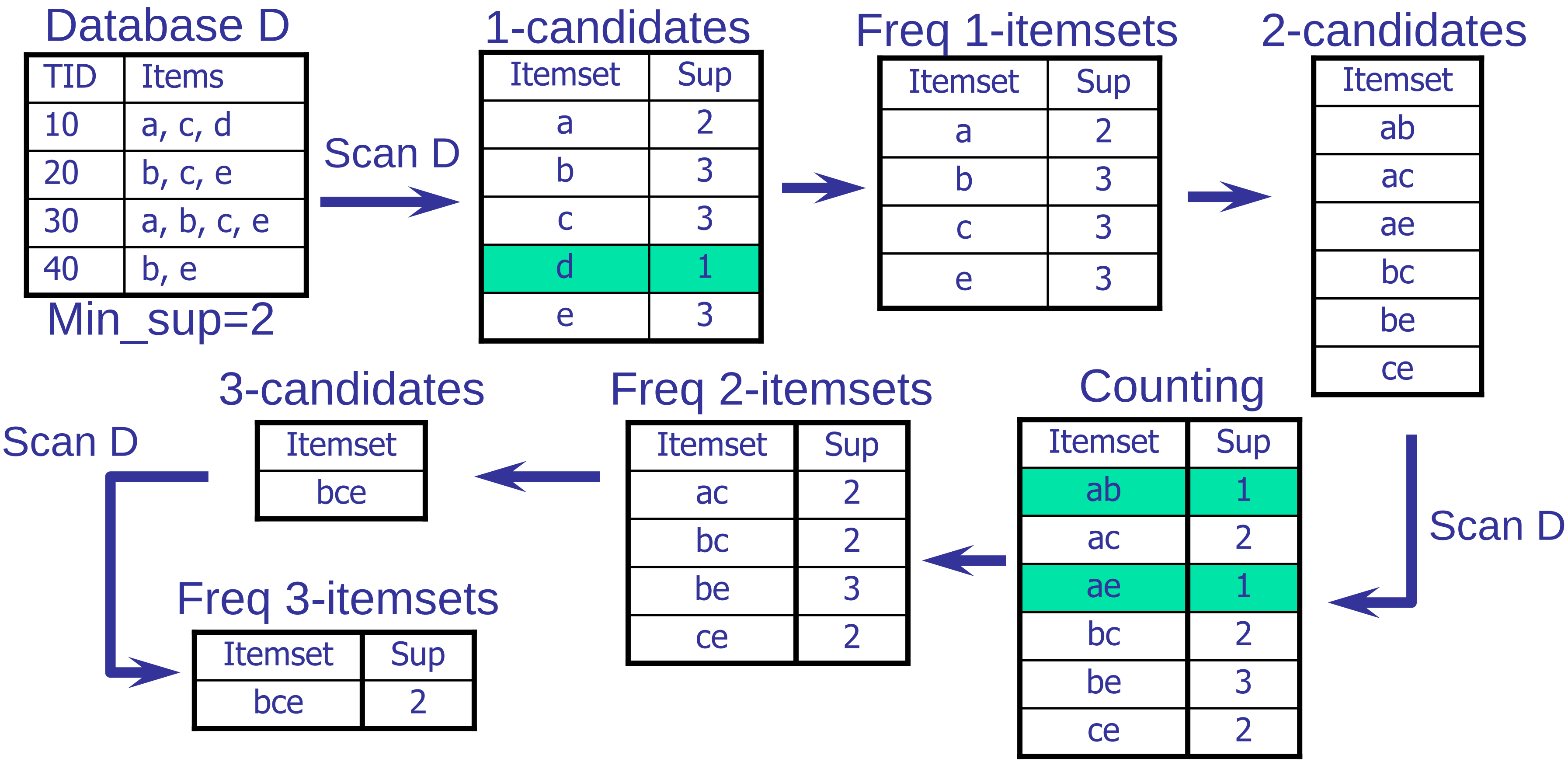


Candidate Itemsets:

{a,b,c}, {a,b,e}, {a,e,f}, {b,e,f}

Apriori Algorithm

- A level-wise, candidate-generation-and-test approach (Agrawal & Srikant 1994)



Database D

TID	Items
10	a, c, d
20	b, c, e
30	a, b, c, e
40	b, e

Min_sup=2

Scan D

1-candidates

Itemset	Sup
a	2
b	3
c	3
d	1
e	3

Freq 1-itemsets

Itemset	Sup
a	2
b	3
c	3
e	3

2-candidates

Itemset
ab
ac
ae
bc
be
ce

Scan D

Counting

Itemset	Sup
ab	1
ac	2
ae	1
bc	2
be	3
ce	2

Freq 2-itemsets

Itemset	Sup
ac	2
bc	2
be	3
ce	2

3-candidates

Itemset
bce

Scan D

Freq 3-itemsets

Itemset	Sup
bce	2

The Apriori Algorithm

- **Join Step**: C_k is generated by joining $L_{k-1}$with itself
- **Prune Step**: Any (k-1)-itemset that is not frequent cannot be a subset of a frequent k-itemset
- Pseudo-code:

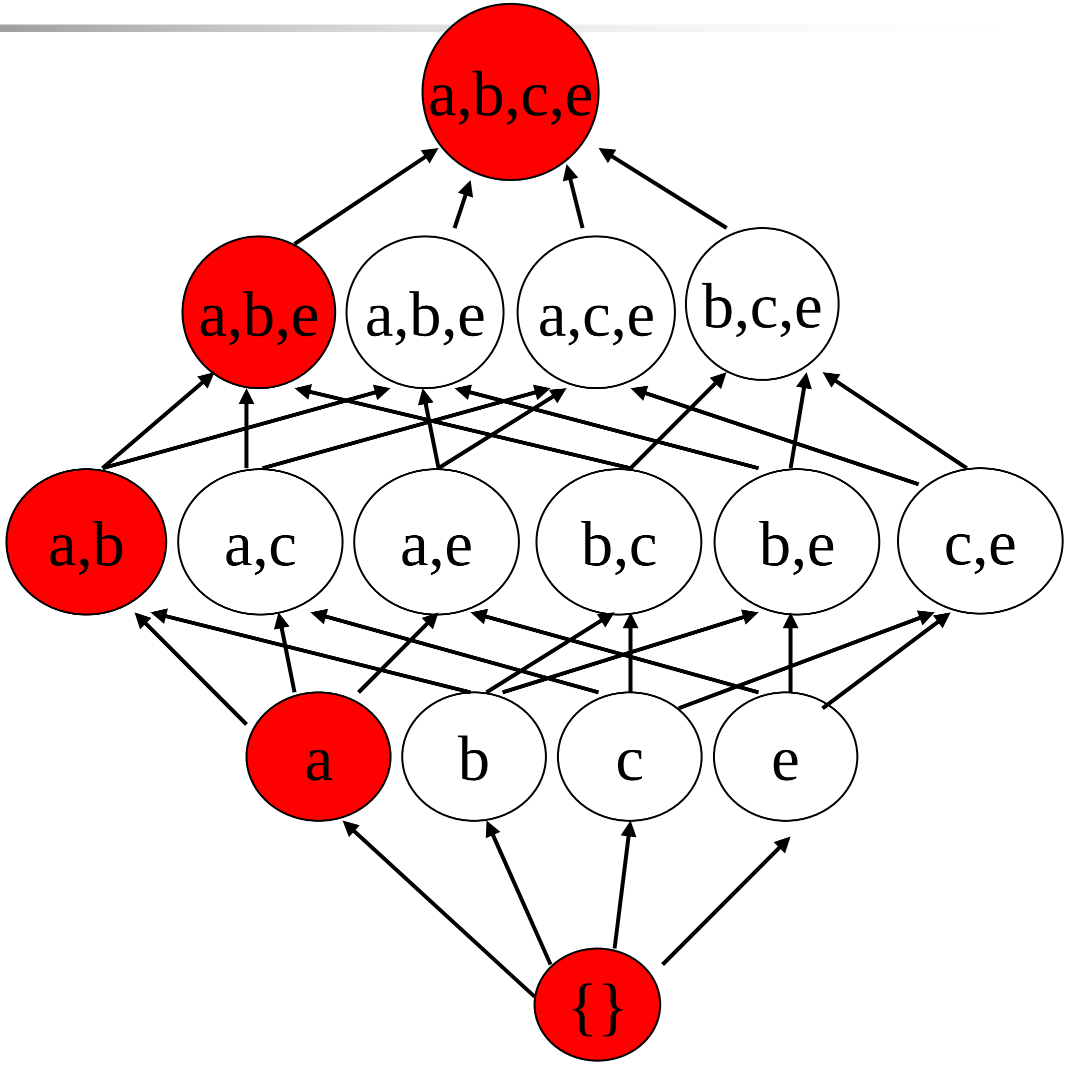
```
$C_k$: Candidate itemset of size k
$L_k$ : frequent itemset of size k

$L_1$ = {frequent items};
for ($k$ = 1; $L_k$ !=$\varnothing$; $k$++) do begin
     $C_{k+1}$ = candidates generated from $L_k$;
    for each transaction $t$ in database do
         increment the count of all candidates in $C_{k+1}$ that are
         contained in $t$
    $L_{k+1}$ = candidates in $C_{k+1}$ with min_support
    end
return $\cup_k L_k$;
```

Rules Generation

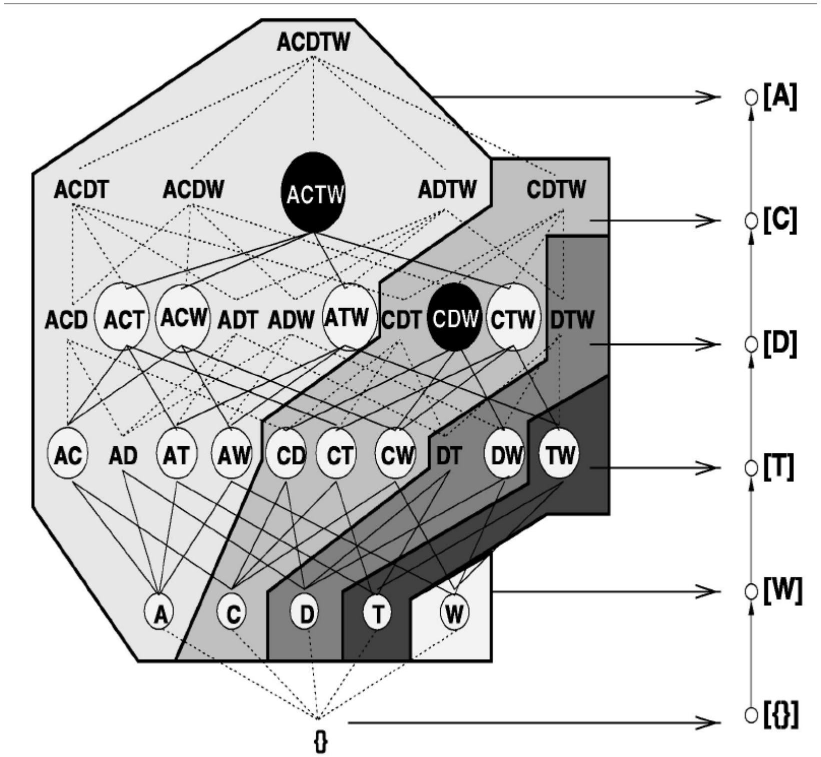
- Since the support of all the frequent itemsets are known, it is possible to derive all rules that satisfied the MINCONF threshold by making use of the support computed.
- Eg. If supp({a,b,c,d})=20 and supp({a,b})=50 then confidence for the rule {a,b=>c,d} is 20/50 = 40%.

Other alternatives

- **Depth-first search**
- Algorithm with **write** or **point**



History of DFS in Frequent Pattern Mining(I)



Equivalence classes when k=1

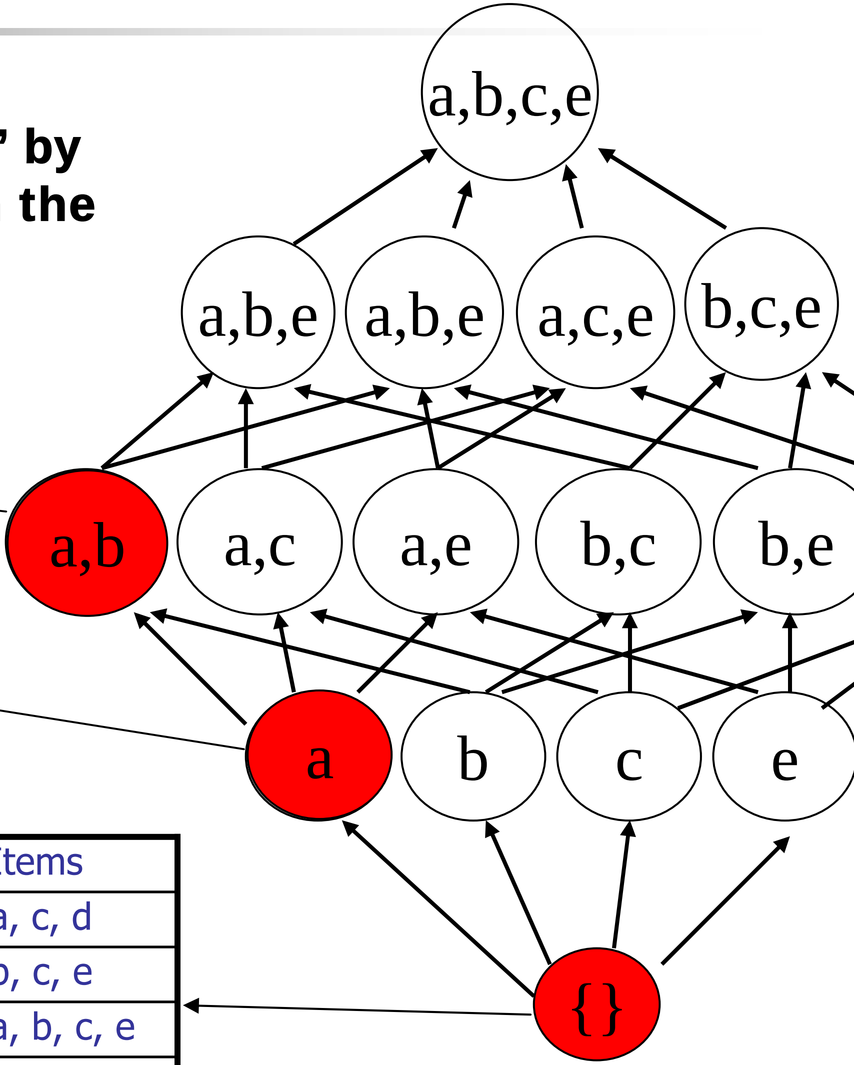
- Concept of equivalence class first introduced in
  - Mohammed J. Zaki, Srinivasan Parthasarathy, Mitsunori Ogihara, Wei Li, "**New Algorithms for Fast Discovery of Association Rules**", *(KDD'97)*, pp 283-286
- "That is, two itemsets are in the same class if they share a common k length prefix. We therefore call k a prefix-based equivalence relation θ_k."
- "The bottom-up search is based on a recursive decomposition of each class into smaller classes induced by the equivalence relation $\theta_{k.}$"
- "The equivalence class lattice can be traversed in either depth-first or breadth-first manner."

History of DFS in Frequent Pattern Mining(II)

- Subsequent work essentially extend the concept of DFS on different types of data, data representation and data structures
- Associative Patterns
  - [SHSB00] VIPER: Implement DFS with bitmap compression of data in vertical format
  - [HaPe00] FP-tree: Implement DFS using a tree structure in horizontal format
- Sequences
  - [Zaki98] SPADE: First to implement DFS for sequential pattern using the prefix equivalence class in vertical format
  - [Pei01] Prefixspan: Implement SPADE using horizontal formal representation
- Data Cube
  - [BeRa99]: BUC: First to implement bottom-up DFS for data cube computation with in-memory points
  - [HPDW01]: Implement BUC using a tree instead of an array

Other alternatives(II): Write-based DFS

Note that we can know the number of occurrence of “ab” by counting the number of “b” in the {a}-projected database !



TID	Items
10	a, c, d
20	b, c, e
30	a, b, c, e
40	b, e

all rows that contain “a”.

{a}-projected database

TID	Items
10	c, d
30	b, c, e

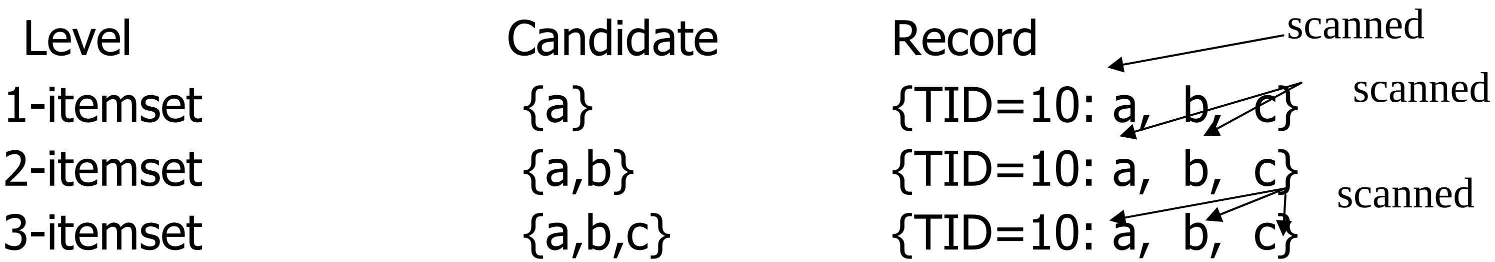
all rows that contain “ab”.

{ab}-projected database

TID	Items
30	c, e

Why would write-based algorithms be faster ?

- Read-based FPM algorithms like Apriori algorithm which will “forget” its previous processing. Eg. let the frequent itemset {a,b,c,d,e} be frequent. Let ‘s look at a record containing {a,b,c}.



Level	Candidate	Record
1-itemset	{a}	{TID=10: a, b, c}
2-itemset	{a,b}	{TID=10: a, b, c}
3-itemset	{a,b,c}	{TID=10: a, b, c}

- Write-based FPM avoid this repeated scanning by doing depth first search

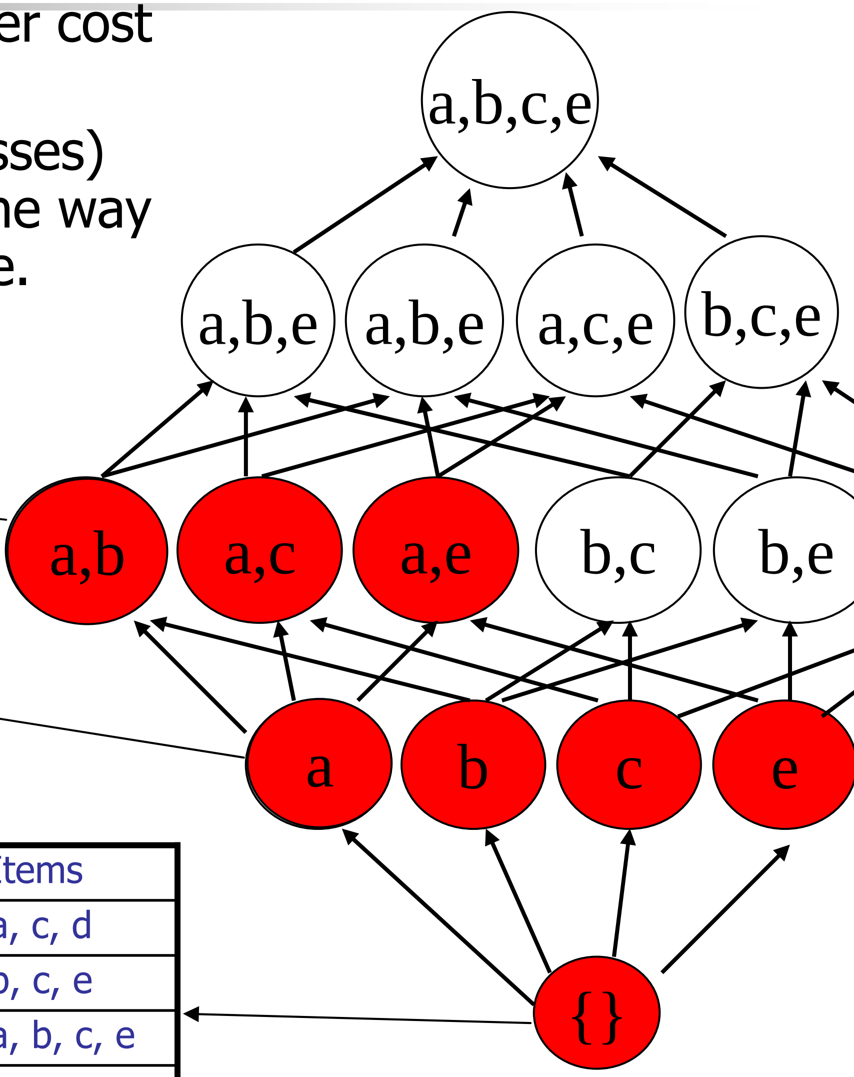
Level	Candidate	Record
1-itemset	{a}	{TID=10: a, b, c}
2-itemset	{a,b}	{TID=10: b, c}
3-itemset	{a,b,c}	{TID=10: c}

Why would write-based algorithms be faster (II)?

- Using storage to reduce run time complexity
- Let us see the counting process as a join algorithm
  - A set of patterns
  - A set of tuples
  - Compare tuples to patterns to see which patterns are found in which tuples
- DFS reduce the set of patterns and tuples being compared i.e. patterns will only be compared against tuples with the same prefix

However, no free lunch……

- Write-based algorithm usually pay a higher cost in term of space complexity.
- Projected databases (i.e. equivalence classes) are materialize for all the red circles on the way to getting the **{a,b} projected** database.



TID	Items
30	c, e

all rows that contain “ab”.

{ab}-projected database

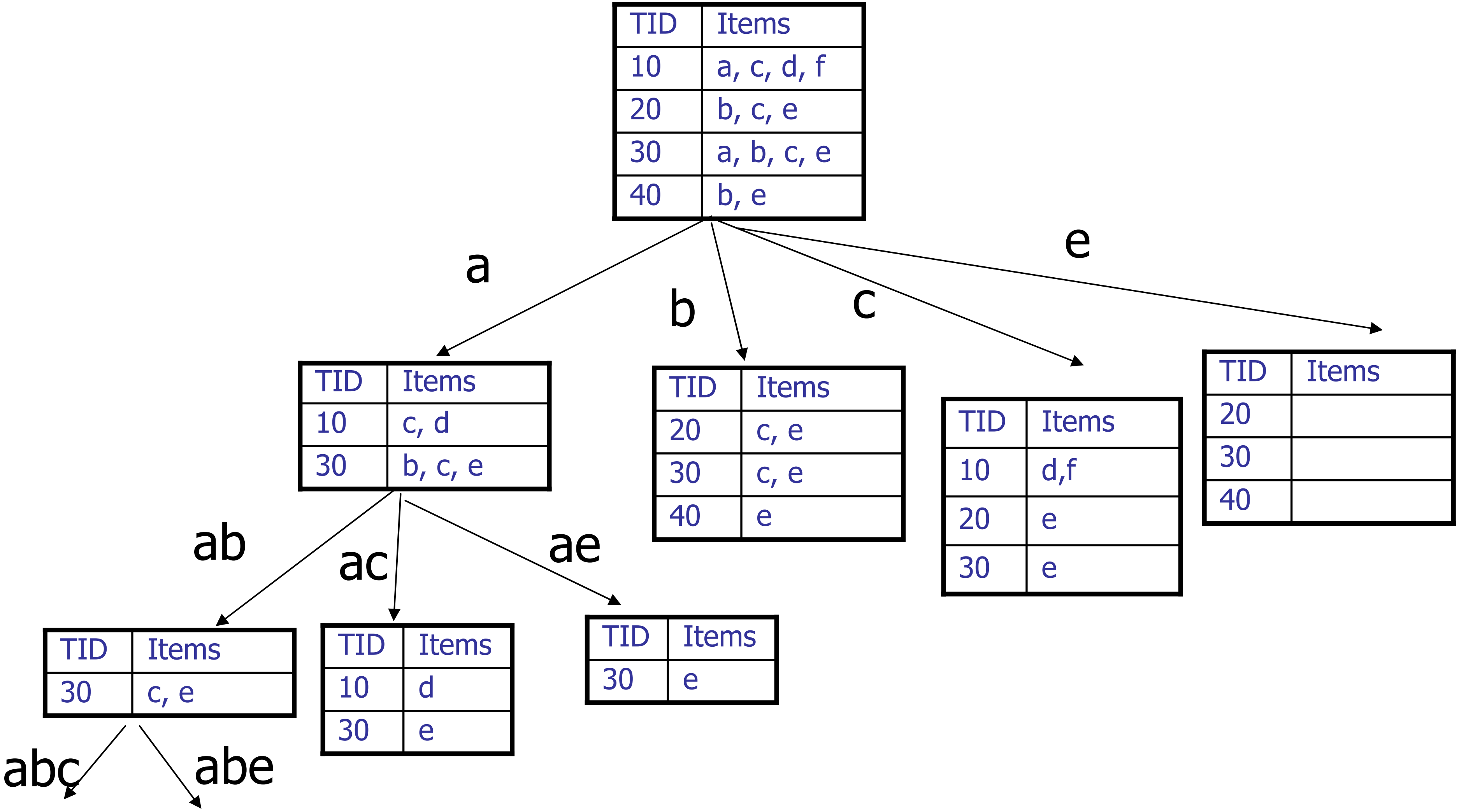
TID	Items
10	c, d
30	b, c, e

all rows that contain “a”.

{a}-projected database

TID	Items
10	a, c, d
20	b, c, e
30	a, b, c, e
40	b, e

No free lunch (II)



TID	Items
10	a, c, d, f
20	b, c, e
30	a, b, c, e
40	b, e

a

TID	Items
10	c, d
30	b, c, e

b

TID	Items
20	c, e
30	c, e
40	e

c

TID	Items
10	d,f
20	e
30	e

e

TID	Items
20	
30	
40	

ab

TID	Items
30	c, e

ac

TID	Items
10	d
30	e

ae

TID	Items
30	e

abc

abe

No candidates ?

- There are claims that write-based algorithm are faster because they don’t generate candidates. This is not true in general. Example, if minsup is 30%, then all the items in the {a}-projected database are false candidates ! (i.e. {a,b}, {a,c}, {a,d}, {a,e}, {a,f} and {a,g} are never frequent

If there is really a length 100 pattern as stated in the FP-tree paper, then the FPgrowth algorithm will visit 2^100 lattice nodes anyway. But only at different point in time!

TID	Items
10	a, c, d
20	b, c, e
30	a, b, c, e
40	a, f
50	b,c
60	a,g
70	f, g,
80	e, g
90	a,b ,g
100	b, e

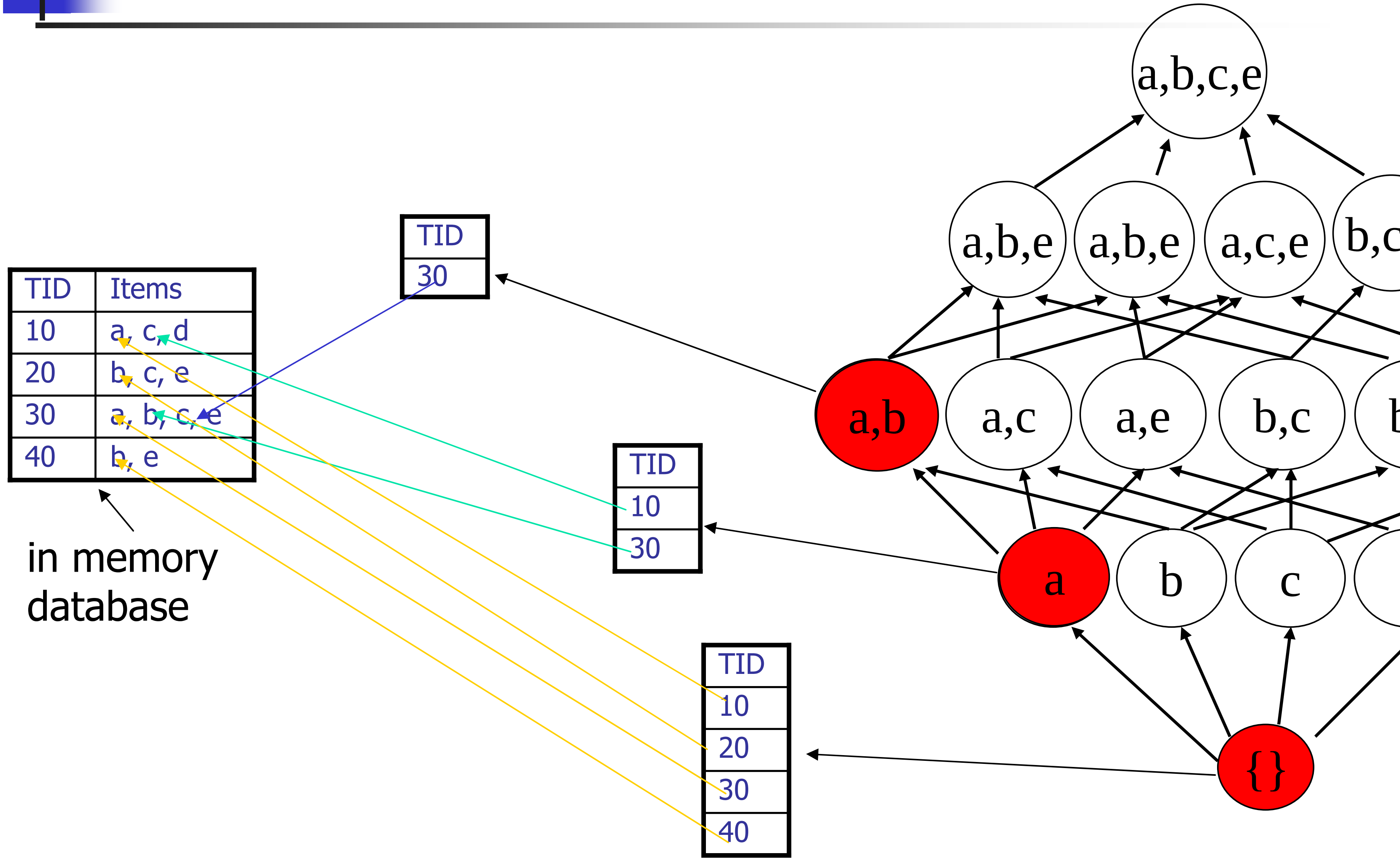
{a}-projection

TID	Items
10	c, d
30	b, c, e
40	f
60	g
90	b, g

Point-based Algorithm (I)

- Same as write-based, but use pointers to the data records in order to avoid physically replicating a part of the database.
- Have to assume that the whole database can be fitted into memory.

Point-based Algorithm (II)



TID	Items
10	a, c, d
20	b, c, e
30	a, b, c, e
40	b, e

in memory database

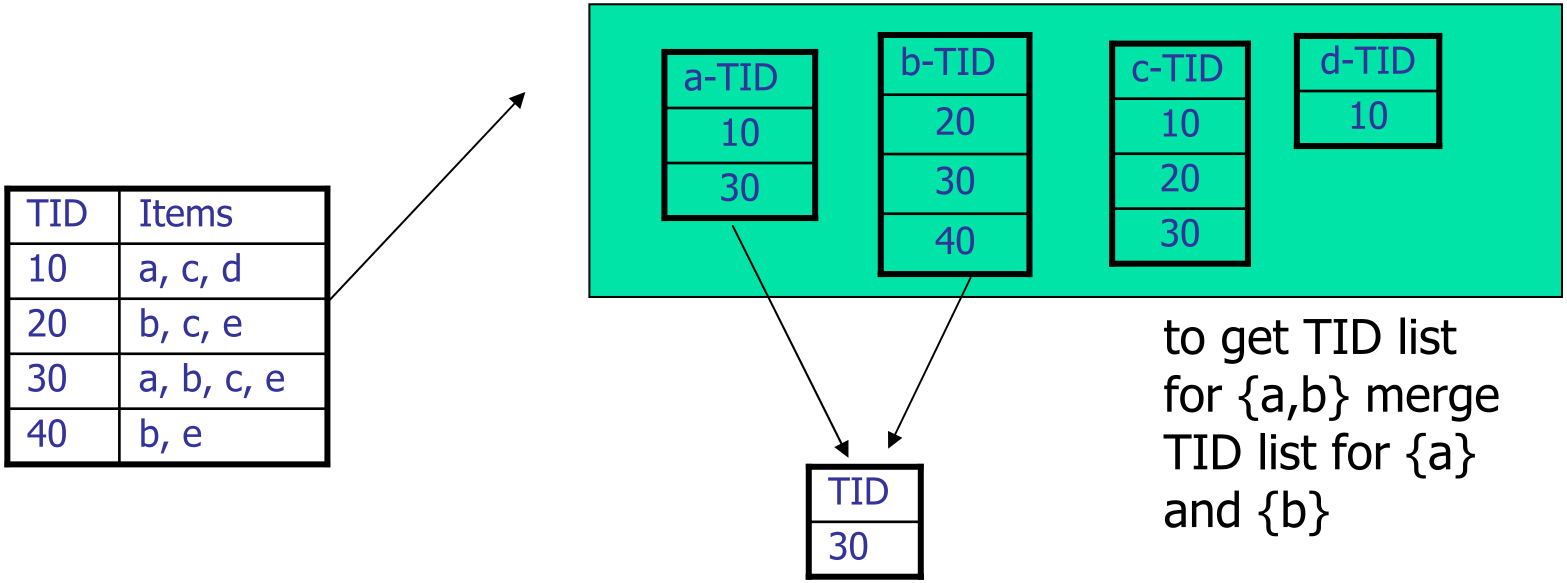
TID
30

TID
10
30

TID
10
20
30
40

Alternative format for write-based FPM

- Horizontal format vs vertical format
  - example shown earlier for write-based FPM is in horizontal format
- Vertical format



TID	Items
10	a, c, d
20	b, c, e
30	a, b, c, e
40	b, e

a-TID
10
30

b-TID
20
30
40

c-TID
10
20
30

d-TID
10

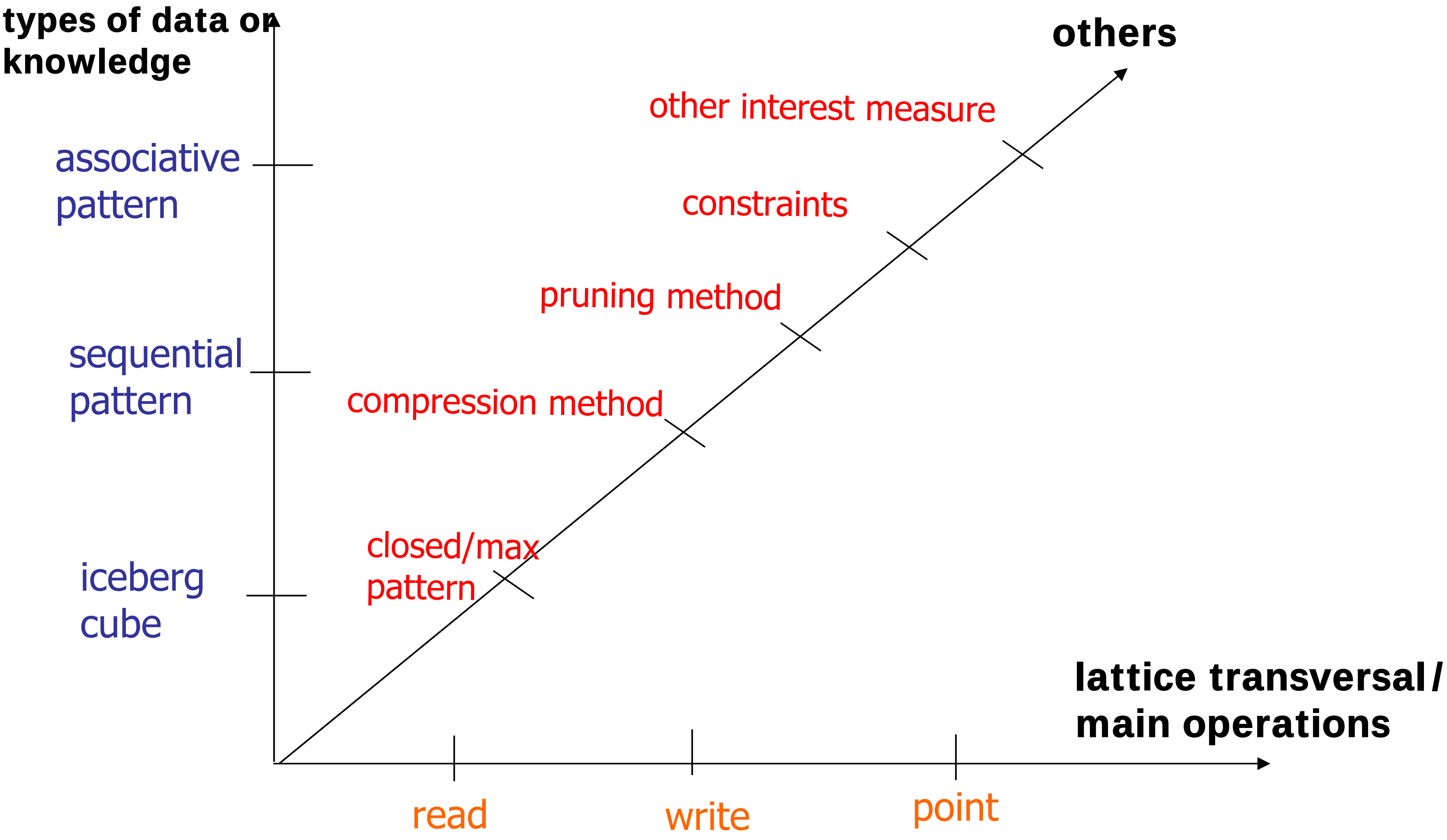
TID
30

to get TID list for {a,b} merge TID list for {a} and {b}

What we covered so far

	Read-based	Write-based	Point-based
Association Mining	Apriori[AgSr94]	Eclat, MaxClique[Zaki01], FPGrowth [HaPe00]	
Sequential Pattern Discovery	GSP[AgSr96]	SPADE [Zaki98,Zaki01], PrefixSpan [PHPC01]	Hmine
Iceberg Cube	Apriori[AgSr94]		BUC[BeRa99], H-cubing [HPDW01]

A Multidimensional View of Frequent Patten Discovery



What we covered so far

	Read-based	~~Write-based~~	Point-based
Association Mining	Apriori[AgSr94]	Eclat, MaxClique[Zaki01], FPGrowth [HaPe00]	Hmine
Sequential Pattern Discovery	GSP[AgSr96]	SPADE [Zaki98,Zaki01], PrefixSpan [PHPC01]	
Iceberg Cube	Apriori[AgSr94]		BUC[BeRa99], H-cubing [HPDW01]

Sequence Databases and Sequential Pattern Analysis

- (Temporal) order is important in many situations
  - Time-series databases and sequence databases
  - Frequent patterns → (frequent) sequential patterns
- Applications of sequential pattern mining
  - Customer shopping sequences:
    - First buy computer, then CD-ROM, and then digital camera, within 3 months.
  - Medical treatment, natural disasters (e.g., earthquakes), science & engineering processes, stocks and markets, telephone calling patterns, Weblog click streams, DNA sequences and gene structures

What Is Sequential Pattern Mining?

- Given a set of sequences, find the complete set of frequent subsequences

A *sequence database*

SID	sequence
10	<a(abc)(ac)d(cf)>
20	<(ad)c(bc)(ae)>
30	<(ef)(ab)(df)cb>
40	<eg(af)cbc>

A *sequence* : < (ef) (ab) (df) c b >

An element may contain a set of items. Items within an element are unordered and we list them alphabetically.

<a(bc)dc> is a *subsequence* of <a(abc)(ac)d(cf)>

Given *support threshold* *min_sup* =2, <(ab)c> is a *sequential pattern*

A Basic Property of Sequential Patterns: Apriori

- Apriori property in sequential patterns
  - If a sequence S is infrequent, then none of the super-sequences of S is frequent
  - E.g, <hb> is infrequent → so do <hab> and <(ah)b>

Given <u>*support threshold*</u> *min_sup* =2

Seq-id	Sequence
10	<(bd)cb(ac)>
20	<(bf)(ce)b(fg)>
30	<(ah)(bf)abf>
40	<(be)(ce)d>
50	<a(bd)bcb(ade)>

GSP

- GSP (Generalized Sequential Pattern) mining
- Outline of the method
  - Initially, every item in DB is a candidate of length-1
  - For each level (i.e., sequences of length-k) do
    - Scan database to collect support count for each candidate sequence
    - Generate candidate length-(k+1) sequences from length-k frequent sequences using Apriori
  - Repeat until no frequent sequence or no candidate can be found
- Major strength: Candidate pruning by Apriori

Finding Length-1 Sequential Patterns

- Initial candidates
  - <a>, <b>, <c>, <d>, <e>, <f>, <g>, <h>
- Scan database once
  - count support for candidates

min_sup =2

Seq-id	Sequence
10	<(bd)cb(ac)>
20	<(bf)(ce)b(fg)>
30	<(ah)(bf)abf>
40	<(be)(ce)d>
50	<a(bd)bcb(ade)>

Cand	Sup
<a>	**3**
<b>	**5**
<c>	**4**
<d>	**3**
<e>	**3**
<f>	**2**
~~<g>~~	1
~~<h>~~	1

Generating Length-2 Candidates

51 length-2 Candidates

	<a>	<b>	<c>	<d>	<e>	<f>
<a>	<aa>	<ab>	<ac>	<ad>	<ae>	<af>
<b>	<ba>	<bb>	<bc>	<bd>	<be>	<bf>
<c>	<ca>	<cb>	<cc>	<cd>	<ce>	<cf>
<d>	<da>	<db>	<dc>	<dd>	<de>	<df>
<e>	<ea>	<eb>	<ec>	<ed>	<ee>	<ef>
<f>	<fa>	<fb>	<fc>	<fd>	<fe>	<ff>

	<a>	<b>	<c>	<d>	<e>	<f>
<a>		<(ab)>	<(ac)>	<(ad)>	<(ae)>	<(af)>
<b>			<(bc)>	<(bd)>	<(be)>	<(bf)>
<c>				<(cd)>	<(ce)>	<(cf)>
<d>					<(de)>	<(df)>
<e>						<(ef)>
<f>						

Without Apriori property, 8*8+8*7/2=92 candidates

Apriori prunes 44.57% candidates

Generating Length-3 Candidates and Finding Length-3 Patterns

- Generate Length-3 Candidates
  - Self-join length-2 sequential patterns
    - <ab>, <aa> and <ba> are all length-2 sequential patterns → <aba> is a length-3 candidate
    - <(bd)>, <bb> and <db> are all length-2 sequential patterns → <(bd)b> is a length-3 candidate
  - 46 candidates are generated
- Find Length-3 Sequential Patterns
  - Scan database once more, collect support counts for candidates
  - 19 out of 46 candidates pass support threshold

The GSP Mining Process

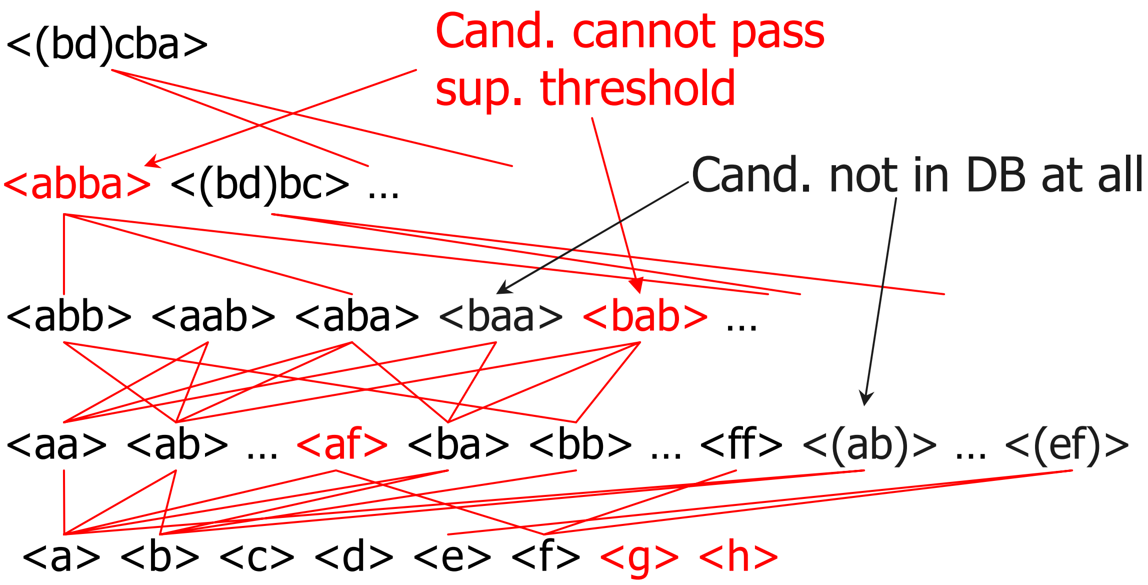
5th scan: 1 cand. 1 length-5 seq. pat.

4th scan: 8 cand. 6 length-4 seq. pat.

3rd scan: 46 cand. 19 length-3 seq. pat. 20 cand. not in DB at all

2nd scan: 51 cand. 19 length-2 seq. pat. 10 cand. not in DB at all

1st scan: 8 cand. 6 length-1 seq. pat.



Cand. cannot pass sup. threshold

Cand. not in DB at all

<(bd)cba>

<abba> <(bd)bc> …

<abb> <aab> <aba> <baa> <bab> …

<aa> <ab> … <af> <ba> <bb> … <ff> <(ab)> … <(ef)>

<a> <b> <c> <d> <e> <f> <g> <h>

min_sup =2

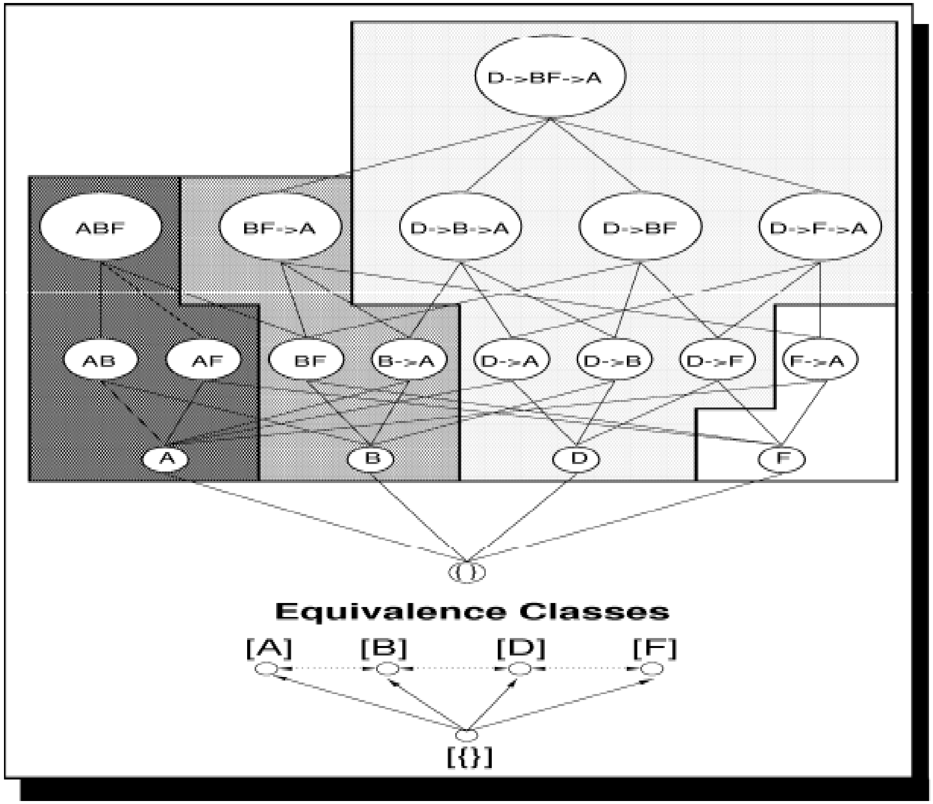
Seq-id	Sequence
10	<(bd)cb(ac)>
20	<(bf)(ce)b(fg)>
30	<(ah)(bf)abf>
40	<(be)(ce)d>
50	<a(bd)bcb(ade)>

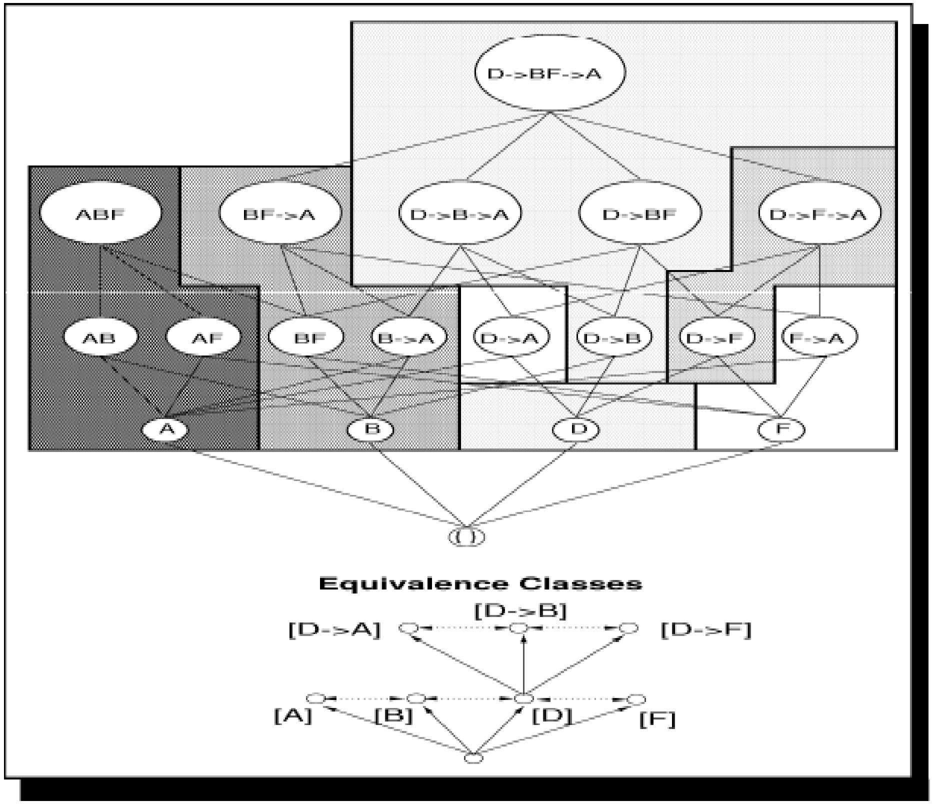
What we covered so far

	Read-based	Write-based	Point-based
Association Mining	Apriori[AgSr94]	Eclat, MaxClique[Zaki01], FPGrowth [HaPe00]	Hmine
Sequential Pattern Discovery	GSP[AgSr96]	SPADE [Zaki98,Zaki01], PrefixSpan [PHPC01]	
Iceberg Cube	Apriori[AgSr94]		BUC[BeRa99], H-cubing [HPDW01]

SPADE

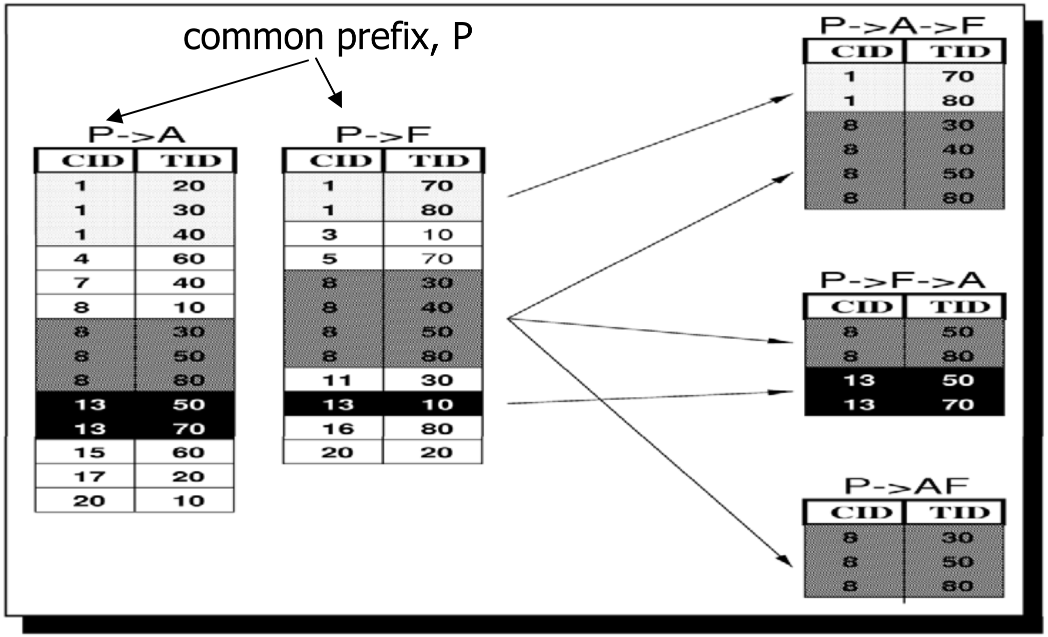
- [Zaki98] Mohammed J. Zaki, "**Efficient Enumeration of Frequent Sequences**", 7th International Conference on Information and Knowledge Management, pp 68-75, Washington DC, November 1998.
- First paper to do depth first search for sequences. Equivalence class based on prefix





SPADE(II)

- Perform both DFS and BFS using Vertical format representation



P->A

CID	TID
1	20
1	30
1	40
4	60
7	40
8	10
8	30
8	50
8	80
13	50
13	70
15	60
17	20
20	10

P->F

CID	TID
1	70
1	80
3	10
5	70
8	30
8	40
8	50
8	80
11	30
13	10
16	80
20	20

P->A->F

CID	TID
1	70
1	80
8	30
8	40
8	50
8	80

P->F->A

CID	TID
8	50
8	80
13	50
13	70

P->AF

CID	TID
8	30
8	50
8	80

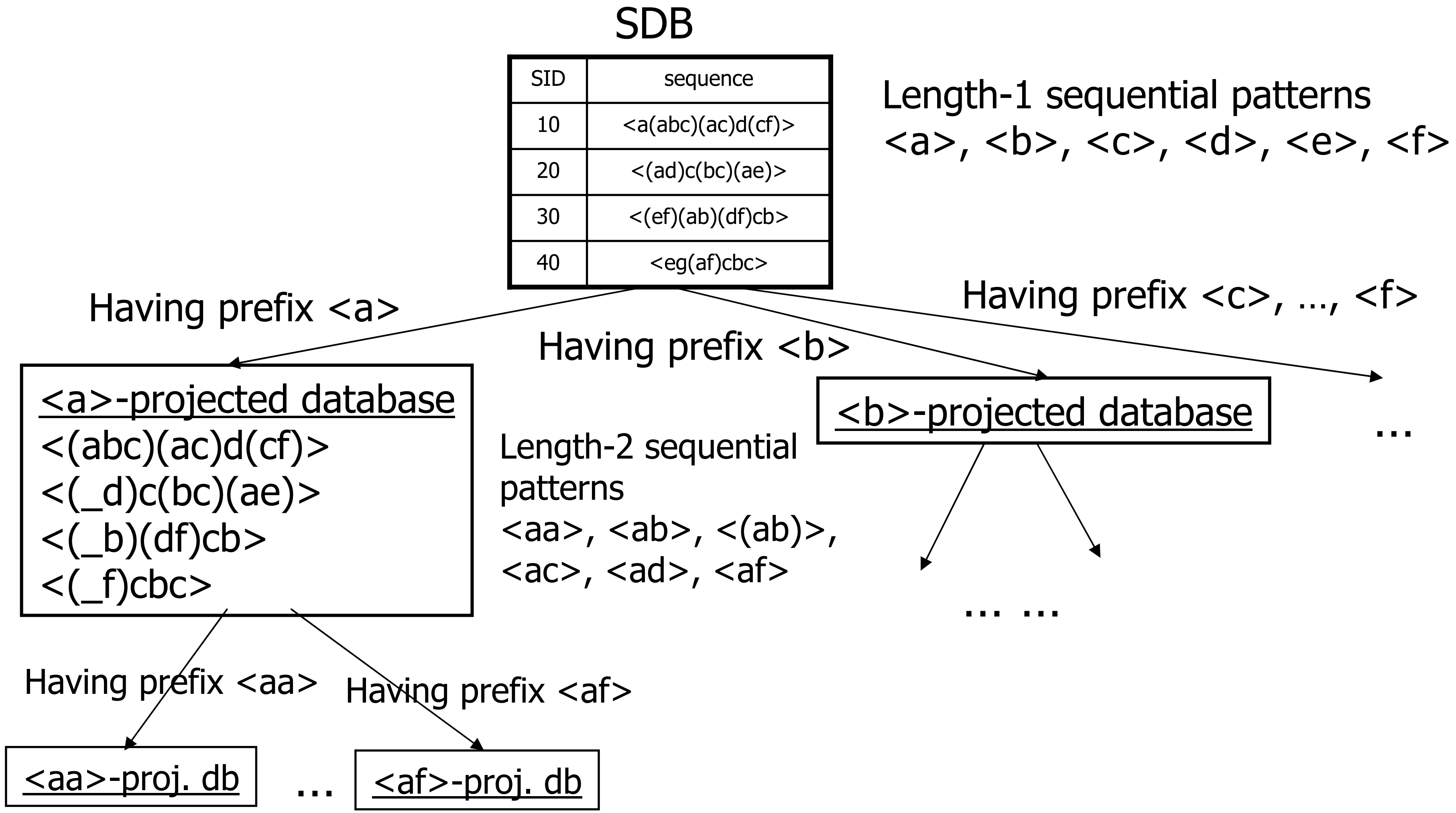
DATABASE

Customer-Id	Transaction-Time	Items
1	10	C D
1	15	A B C
1	20	A B F
1	25	A C D F
2	15	A B F
2	20	E
3	10	A B F
4	10	D G H
4	20	B F
4	25	A G H

A		B		D		F	
CID	TID	CID	TID	CID	TID	CID	TID
1	15	1	15	1	10	1	20
1	20	1	20	1	25	1	25
1	25	2	15	4	10	2	15
2	15	3	10			3	10
3	10	4	20			4	20
4	25						

Vertical format representation

PrefixSpan: Essentially Implement SPADE using Horizontal Format



SDB

SID	sequence
10	<a(abc)(ac)d(cf)>
20	<(ad)c(bc)(ae)>
30	<(ef)(ab)(df)cb>
40	<eg(af)cbc>

Length-1 sequential patterns
<a>, <b>, <c>, <d>, <e>, <f>

Having prefix <a>

Having prefix <b>

Having prefix <c>, …, <f>

<a>-projected database
<(abc)(ac)d(cf)>
<(_d)c(bc)(ae)>
<(_b)(df)cb>
<(_f)cbc>

Length-2 sequential patterns
<aa>, <ab>, <(ab)>,
<ac>, <ad>, <af>

<b>-projected database

…

… …

Having prefix <aa>

Having prefix <af>

<aa>-proj. db

…

<af>-proj. db

Iceberg Cubes

- Let each tuple in the database have certain measure or weight. Ensure that the total weight exceed a certain threshold.

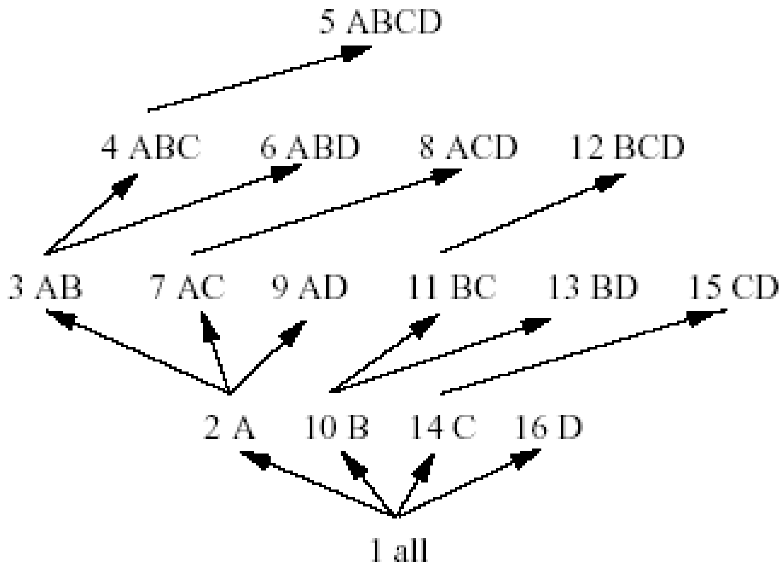
```
CREATE CUBE Sales_Iceberg AS
SELECT month, city, cust_grp,
          TOTAL(price), COUNT(*)
FROM Sales_Infor
CUBEBY month, city, cust_grp
HAVING COUNT(*)>=50 and
TOTAL(price) > 10000
```

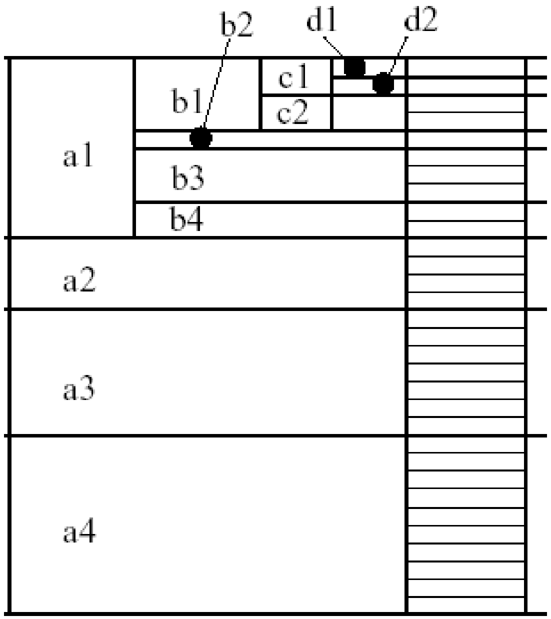
Month	City	Cust_grp	Prod	Cost	Price	Count
Jan	Tor	Edu	Printer	500	485	1
Mar	Van	Edu	HD	540	520	1
…	…	…	…	…	…	

What are we finding if we just limit the count ?

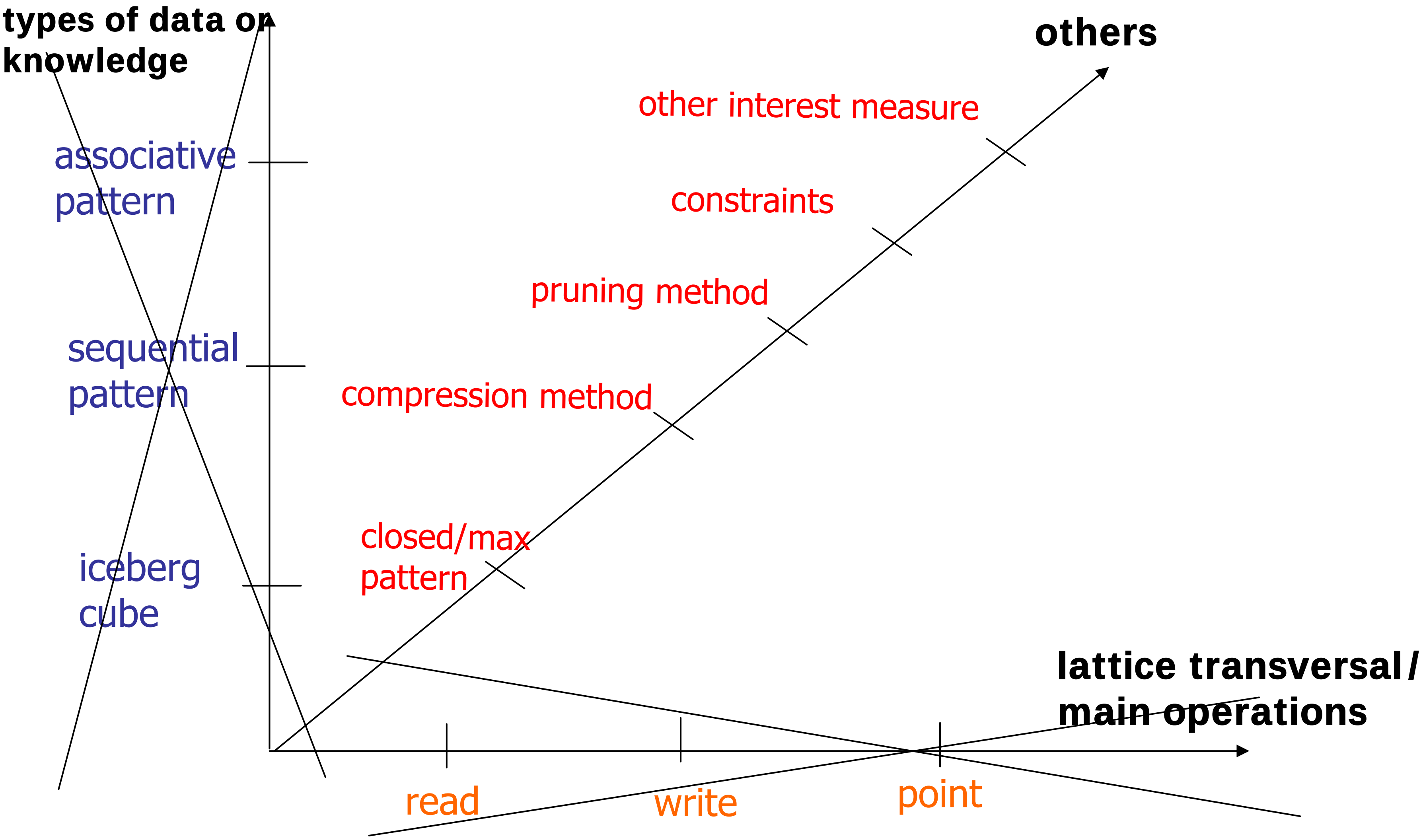
Computing Iceberg Cube

- Compute iceberg queries efficiently by Apriori:
  - First compute lower dimensions
  - Then compute higher dimensions only when all the lower ones are above the threshold
- Pointer-Based: BUC (Bottom-Up Cube Computation)





A Multidimensional View of Frequent Patten Discovery



Efficiency Enhancement

- Methods to Enhance Pruning
  - DHP
  - Partition
  - Sampling
- Compression Method
  - FP-tree
  - VIPER: Vertical Compression

DHP: Reduce the Number of Candidates

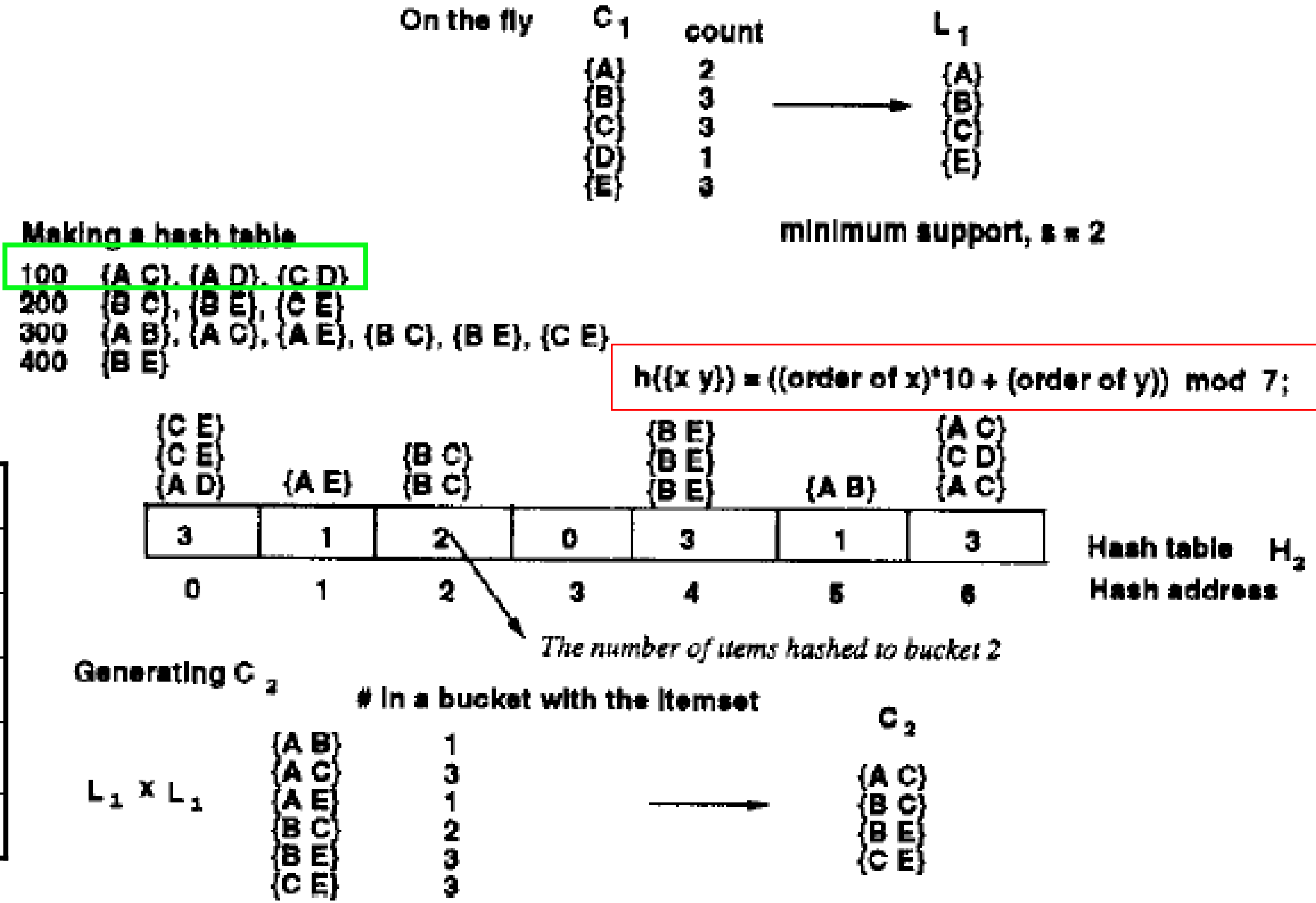
- Hash candidate itemsets into a hash table.
- A hashing bucket count <min_sup → every candidate in the buck is infrequent
- “An Effective Hash-Based Algorithm for Mining Association Rules”, J. Park, M. Chen, and P. Yu, 1995

DHP: An example

Database *D*

TID	Items
100	A C D
200	B C E
300	A B C E
400	B E

Items	Order
A	1
B	2
C	3
D	4
E	5



On the fly

C_1	count
{A}	2
{B}	3
{C}	3
{D}	1
{E}	3

L_1: {A} {B} {C} {E}

minimum support, s = 2

Making a hash table

100 {A C}, {A D}, {C D}
200 {B C}, {B E}, {C E}
300 {A B}, {A C}, {A E}, {B C}, {B E}, {C E}
400 {B E}

h({x y}) = ((order of x)*10 + (order of y)) mod 7;

Hash address	0	1	2	3	4	5	6
Itemsets	{C E} {C E} {A D}	{A E}	{B C} {B C}		{B E} {B E} {B E}	{A B}	{A C} {C D} {A C}
Hash table H_2	3	1	2	0	3	1	3

The number of uems hashed to bucket 2

Generating C_2

$L_1 \times L_1$	# in a bucket with the itemset
{A B}	1
{A C}	3
{A E}	1
{B C}	2
{B E}	3
{C E}	3

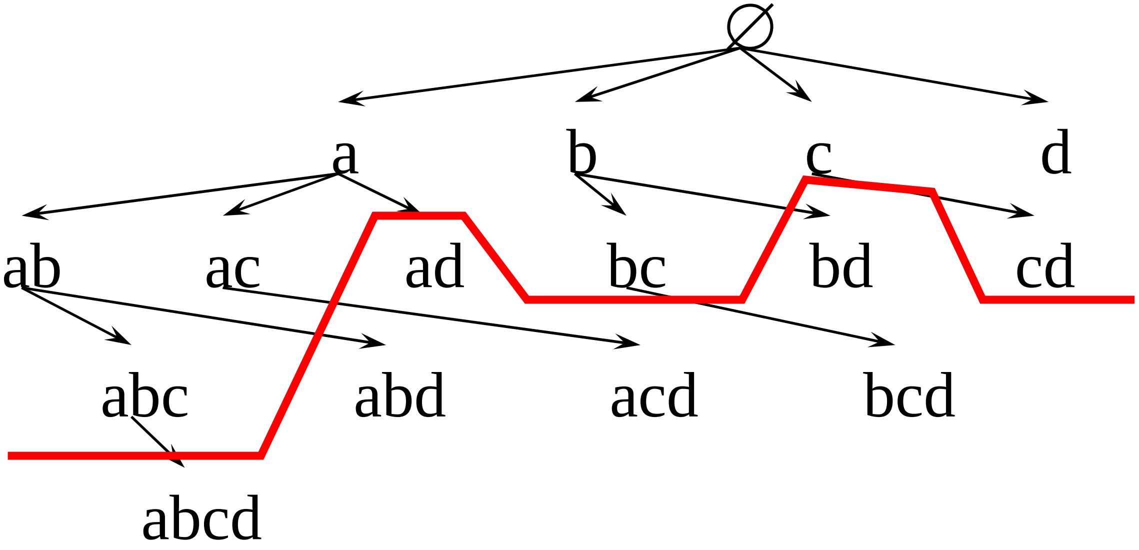
C_2: {A C} {B C} {B E} {C E}

Partition: Scan Database Only Twice

- Partition the database into n partitions
- Itemset X is frequent → X frequent in at least one partition
  - Scan 1: partition database and find local frequent patterns (support > minsup/n)
  - Scan 2: consolidate global frequent patterns
- A. Savasere, E. Omiecinski, and S. Navathe, 1995

Borders of Frequent Itemsets

- Connected
  - X and Y are frequent and X is an ancestor of Y → all patterns between X and Y are frequent

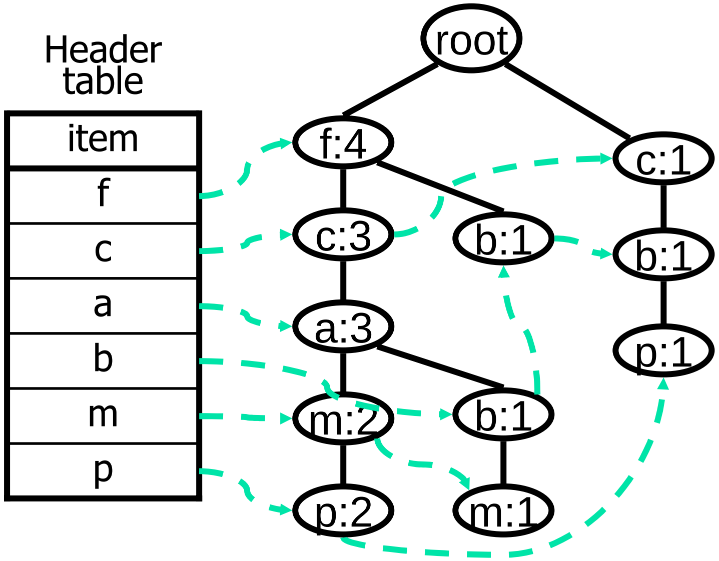


Sampling for Frequent Patterns

- Select a sample of original database, mine frequent patterns within sample using Apriori
- Scan database once to verify frequent itemsets found in sample, only borders of closure of frequent patterns are checked
  - Example: check abcd instead of ab, ac, …, etc.
- Scan database again to find missed frequent patterns

Compress Database by FP-tree

- 1st scan: find freq items
  - Only record freq items in FP-tree
  - F-list: f-c-a-b-m-p
- 2nd scan: construct tree
  - Order freq items in each transaction w.r.t. f-list
  - Explore sharing among transactions



TID	Items bought	(ordered) freq items
100	f, a, c, d, g, I, m, p	f, c, a, m, p
200	a, b, c, f, l,m, o	f, c, a, b, m
300	b, f, h, j, o	f, b
400	b, c, k, s, p	c, b, p
500	a, f, c, e, l, p, m, n	f, c, a, m, p

Find Patterns Having Item “p”

- Only transactions containing p are needed
- Form p-projected database
  - Starting at entry p of header table
  - Follow the side-link of frequent item p
  - Accumulate all transformed prefix paths of p

p-projected database $\text{TDB}|_p$

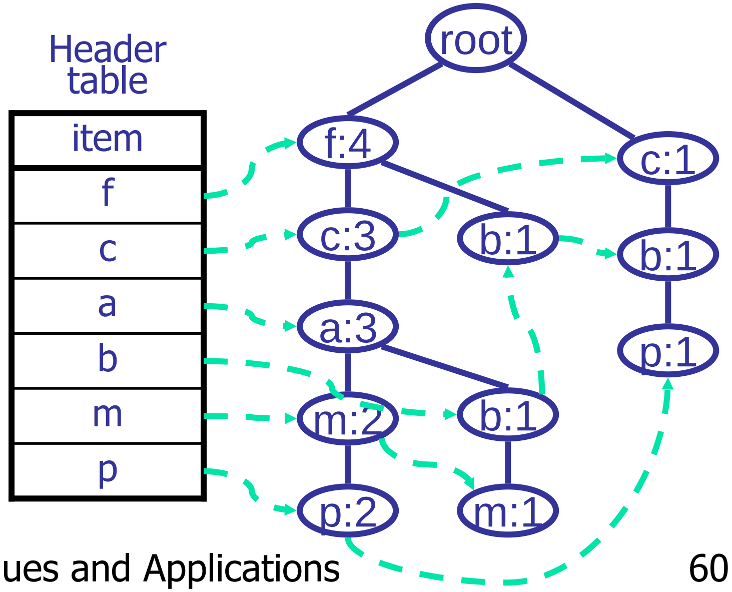
fcam: 2

cb: 1

Local frequent item: c:3

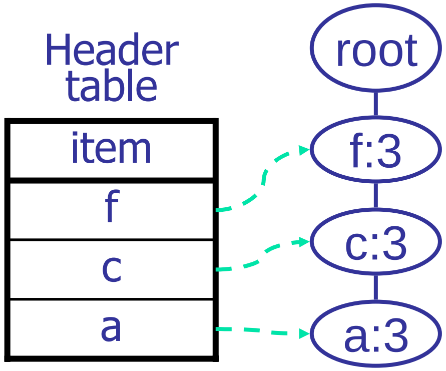
Frequent patterns containing p

p: 3, pc: 3

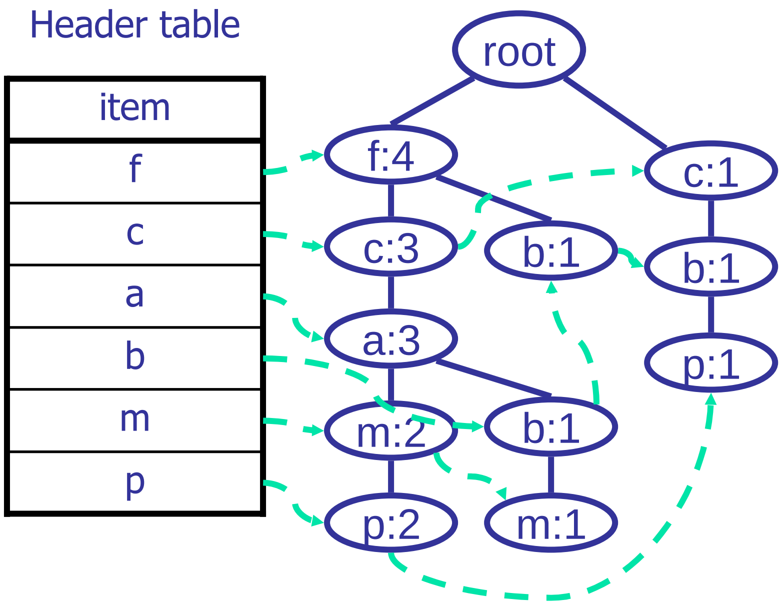


Find Patterns Having Item m But No p

- Form m-projected database TDB|m
  - Item p is excluded
  - Contain fca:2, fcab:1
  - Local frequent items: f, c, a
- Build FP-tree for TDB|m

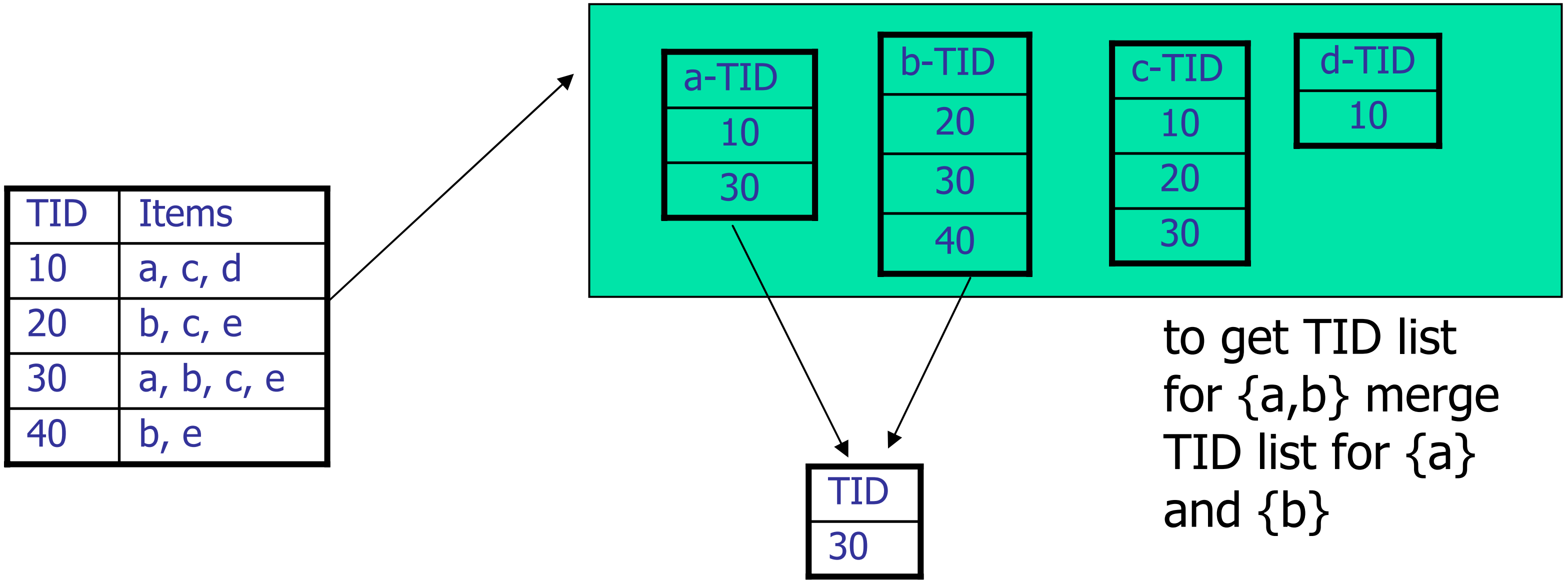


m-projected FP-tree



VIPER: Vertical Itemset Partitioning for Efficient Rule-extraction(I)

- Review:



TID	Items
10	a, c, d
20	b, c, e
30	a, b, c, e
40	b, e

a-TID
10
30

b-TID
20
30
40

c-TID
10
20
30

d-TID
10

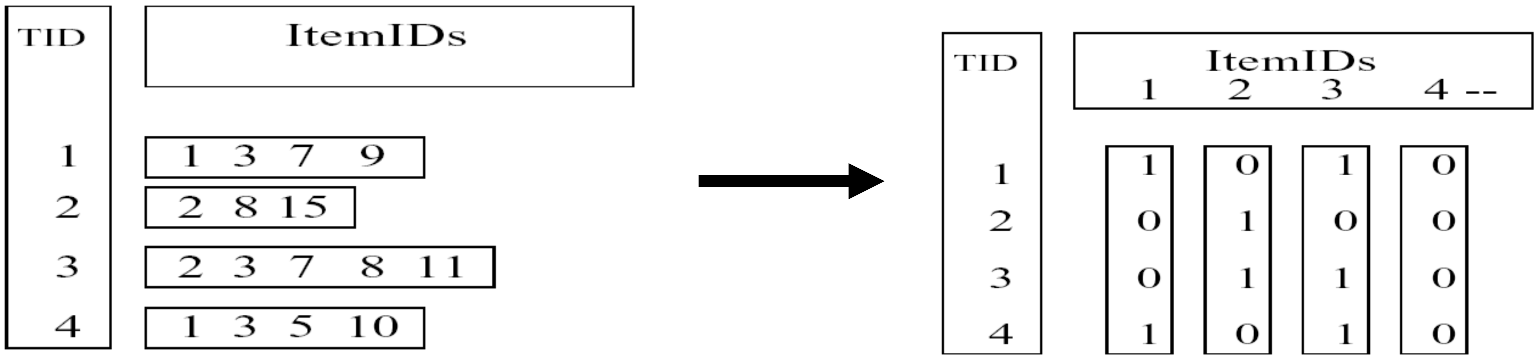
TID
30

to get TID list for {a,b} merge TID list for {a} and {b}

- Problem: TID lists can be large since each TID will take up $\log_2 N$ bits, N=number of transactions.

VIPER: Vertical Itemset Partitioning for Efficient Rule-extraction(II)

- Solution: Use bitmap encoding



TID	ItemIDs
1	1 3 7 9
2	2 8 15
3	2 3 7 8 11
4	1 3 5 10

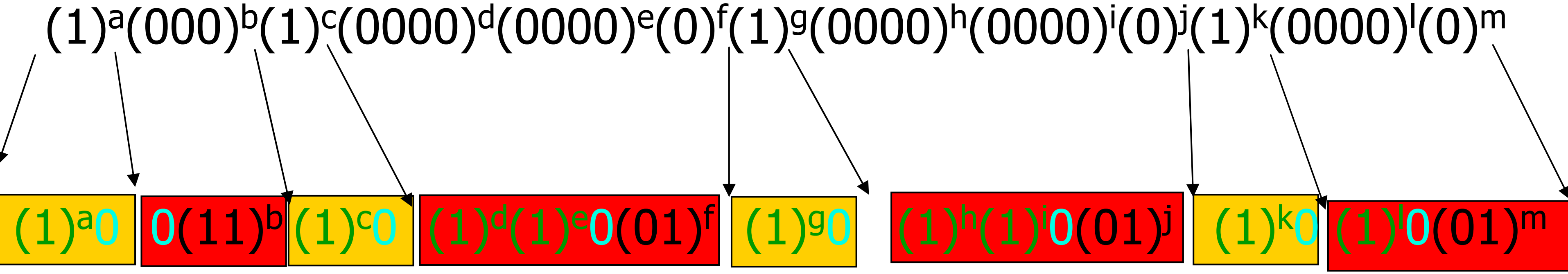
TID	ItemIDs 1	2	3	4 --
1	1	0	1	0
2	0	1	0	0
3	0	1	1	0
4	1	0	1	0

- Golomb encoding scheme
  - Divide runs of 1's and 0's into group of fixed size W_0,W_1
  - Weight bits:Each FULL group represented with a single bit set to "1"
  - Count field: Last partial group represent using $\log_2 W_i$ bits
  - Use "0" as field separator to separate weight bits and count field

VIPER: Vertical Itemset Partitioning for Efficient Rule-extraction(III)

bitmap for an item :1000100000000001000000000100000

- Example of encoding: W1=1, W0=4



$(1)^a(000)^b(1)^c(0000)^d(0000)^e(0)^f(1)^g(0000)^h(0000)^i(0)^j(1)^k(0000)^l(0)^m$

$(1)^a 0$ | $0(11)^b$ | $(1)^c 0$ | $(1)^d(1)^e 0(01)^f$ | $(1)^g 0$ | $(1)^h(1)^i 0(01)^j$ | $(1)^k 0$ | $(1)^l 0(01)^m$

🟨 run of 1s

🟥 run of 0s

1,0: weight bits

1,0: count field

0: field separator

It is shown that such an encoding give better compression than storing TID by 3 times.

VIPER vs FP-tree: A Comparison

- Compression Strategy
  - VIPER compress the bitmap for each item/itemset individually
  - In FP-tree, items at the bottom of the tree might not be compressed. Have to pay memory cost for storing pointers.
- Experiments Conduct
  - VIPER: conducted experiments on dataset up to 1.2GB
  - FP-tree: largest dataset tested on is approximately 5MB
- Locality on Secondary Storage
  - VIPER: each compressed bitmap can be stored consecutively on the disk
  - FP-tree: Don't see an obvious way since each transaction must be grouped together and they will be visited solely to retrieve one item at different point in time

VIPER vs FP-tree(II)

- But are they really different?
  - Can they be converted to the other format easily?
  - Hint: What is the complexity of merging k sorted list?
- Analogy:
  - Normalize vs Un-normalized database
    - One pay a cost to join up tables, the other save the cost of a relational table join but pay in term of storage and when only one of the attributes is needed
  - Row store vs Column Store
    - **Scalable Semantic Web Data Management Using Vertical Partitioning (Best Paper Award)** *Proceedings of VLDB* , 2007. Daniel Abadi, Adam Marcus, Samuel Madden, and Katherine Hollenbach
- My take: Growth in CPU/GPU will mean that vertical format could eventually get rewarded
  - http://db.lcs.mit.edu/madden/high_perf.pdf

Go Green: Recycle and Reuse Frequent Patterns

- Data Mining in fact is a iterative process where mining parameters are adjusted iteratively so as to obtain interesting results. In the case of FPM, we need to adjust minsup to find interesting patterns
- How can the patterns found in one round be useful for the next round of mining?
- What information is available in previous round?
  - Discovered Knowledge: closed patterns for query answering; clustering for sparse or empty region identification
  - Selectivity and Distribution of Data Values: decision tree can provide information for query optimization
  - Mining Cost: data mining plan

Go Green: Recycle and Reuse Frequent Patterns(II)

- Since FP-tree and VIPER work well with simple compression, what if we use more complex compression?
  - [Cong04] Gao Cong, Beng Chin Ooi, Kian-Lee Tan, Anthony K. H. Tung, "**Go Green: Recycle and Reuse Frequent Patterns**". International Conference on Data Engineering (ICDE'2004), Boston, 2004
- How can compression speed up mining?
- Example: old minimum support = 3 and new = 2

id	Items
100	a,c,d,e,f,g
200	b,c,d,f,g
300	c,e,f,g
400	a,c,e,i
500	a,e,h

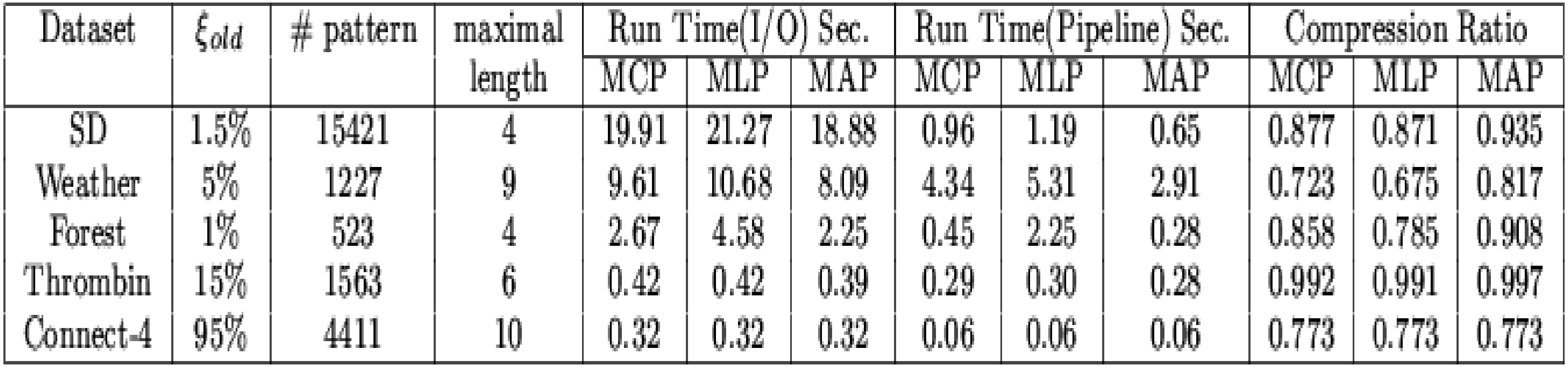
group	Id	Outlying items	Ordered frequent
fgc	100	a,d,e	d, a, e,
	200	b,d	d
	300	e	e
ae	400	c,i	c
	500	h	

Compression Strategies

- Thus, for each transaction, we need to pick one of the discovered patterns to compressed/cover it
- Possible criteria:
  - Strategy 1: Maximal Length Principle (MLP):
    - The utility function is U(X) = |X|*|DB|+X.C, where X.C is the number of tuples that contain pattern X
  - Strategy 2: Maximal Area Principle (MAP)
    - The utility function is U(X) = |X|* X.C
  - Strategy 3: Minimize Cost Principle (MCP)}
    - The utility function is U(X)= $(2^{|X|}-1)$ * X.C
- Each tuple is compressed with a pattern selected based on utility value

Analysis of Compression Strategies

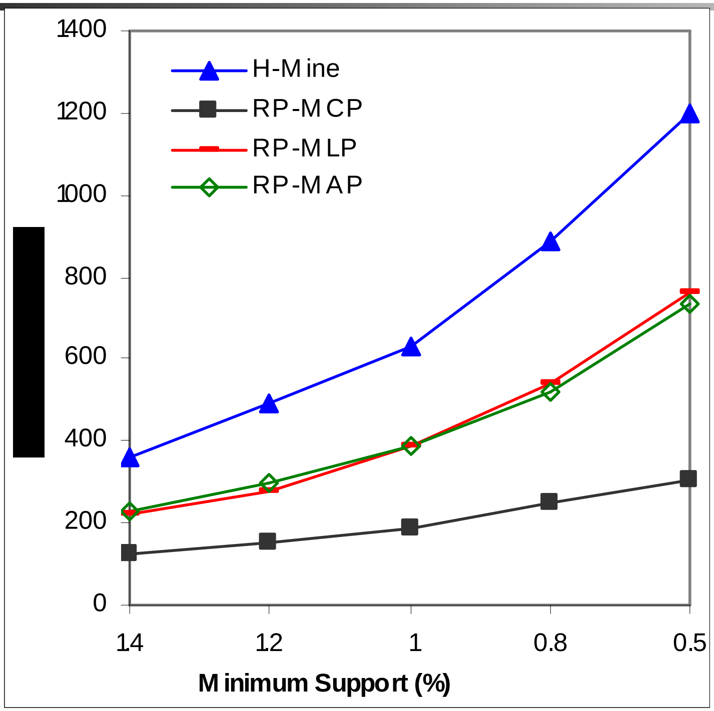
Compression time and compression ratio

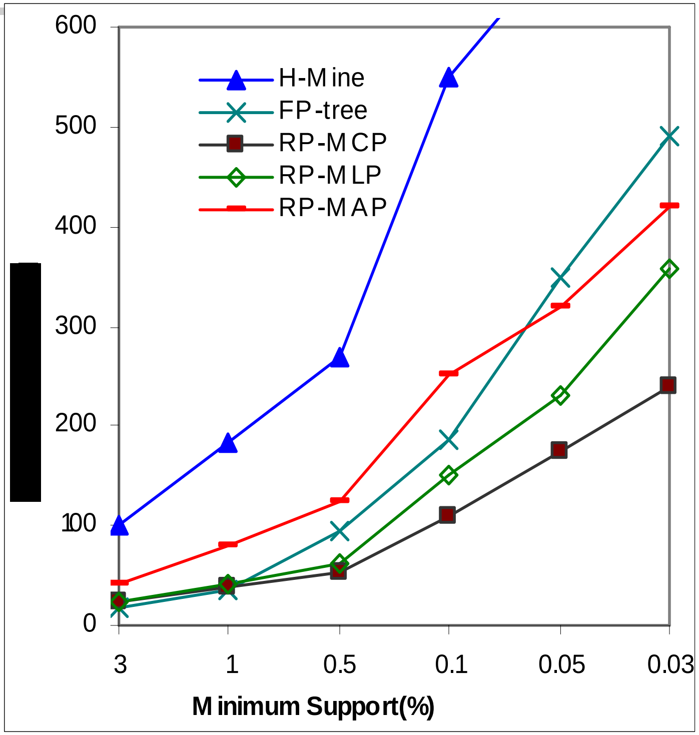


Dataset	ξ_{old}	# pattern	maximal length	Run Time(I/O) Sec.			Run Time(Pipeline) Sec.			Compression Ratio		
				MCP	MLP	MAP	MCP	MLP	MAP	MCP	MLP	MAP
SD	1.5%	15421	4	19.91	21.27	18.88	0.96	1.19	0.65	0.877	0.871	0.935
Weather	5%	1227	9	9.61	10.68	8.09	4.34	5.31	2.91	0.723	0.675	0.817
Forest	1%	523	4	2.67	4.58	2.25	0.45	2.25	0.28	0.858	0.785	0.908
Thrombin	15%	1563	6	0.42	0.42	0.39	0.29	0.30	0.28	0.992	0.991	0.997
Connect-4	95%	4411	10	0.32	0.32	0.32	0.06	0.06	0.06	0.773	0.773	0.773

MLP ≥ MCP ≥ MAP

Example Runtime





Weather

- MCP better than MLP & MAP
- better compression does not necessary means better performance.
- minimizing mining cost (MCP) is more effective than minimizing storage space (MLP & MAP)

Interesting Observation

- When minimum support is low, RP-MCP >> H-Mine
- In fact

time to mine with high support
time for mining with low support >> + compression time
+ mine with low support

- This suggests the possibility that we could divide a new mining task with low minimum support into two steps:
  - (a) We first run it with a high minimum support;
  - (b) Compressing the database with the strategy MCP and mine the compressed database with the required low minimum support.

Extensions

- There are many other ways to find patterns to compress that data
  - Mining on a sample of the data
  - Other pattern mining methods
    - H. V. Jagadish, Raymond T. Ng, Beng Chin Ooi, Anthony K. H. Tung, "ItCompress: An Iterative Semantic Compression Algorithm". International Conference on Data Engineering (ICDE'2004), Boston, 2004
- In fact a simple reordering of transaction based on the patterns they are assigned to can speed up things substantially
  - G. Buehrer, S. Parthasarathy, and A. Ghoting. Out-of-core Frequent Pattern Mining on a Commodity PC. Proceedings of the ACM International Conference on Knowledge Discovery and Data Mining (SIGKDD), 2006, pp. 86-95
- How do we recycle the patterns for vertical format?
- Can patterns be used to speed up other mining/querying tasks?
  - Xifeng Yan, Philip S. Yu, Jiawei Han: Graph Indexing: A Frequent Structure-based Approach. SIGMOD Conference 2004: 335-346

Finding Interesting Patterns

- Correlation/Lift
- Reducing the number of patterns
  - Max pattern
  - Closed pattern
- Applying constraints

Misleading Rules

- Play basketball ⇒ eat cereal [40%, 66.7%]
  - The overall percentage of students eating cereal is 75%, is higher than 66.7%
  - Play basketball ⇒ not eat cereal [20%, 33.3%] is more accurate, though with lower support and confidence

	Basketball	Not basketball	Sum (row)
Cereal	2000	1750	3750
Not cereal	1000	250	1250
Sum(col.)	3000	2000	5000

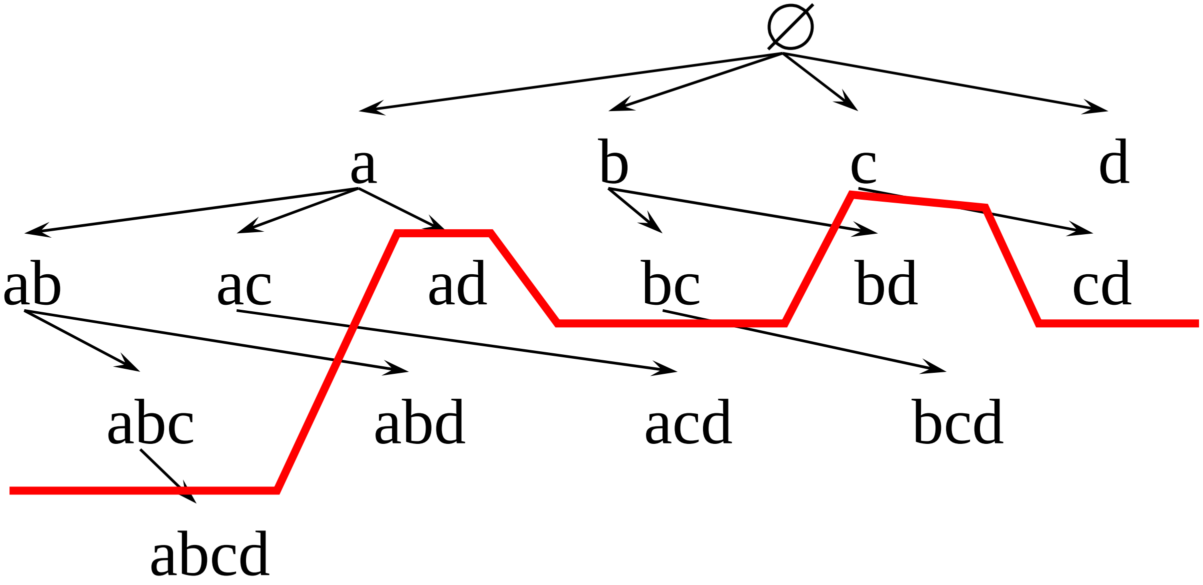
Correlation and Lift

- P(B|A)/P(B) is called the lift of rule A => B

$$corr_{A,B} = \frac{P(A \cup B)}{P(A)P(B)}$$

Borders and Max-patterns

- Max-patterns: borders of frequent patterns
  - All subset of max-pattern is frequent
  - At least one superset of max-pattern is infrequent



MaxMiner: Mining Max-patterns

Tid	Items
10	A,B,C,D,E
20	B,C,D,E,
30	A,C,D,F

Min_sup=2

- 1st scan: find frequent items
  - A, B, C, D, E
- 2nd scan: find support for
  - AB, AC, AD, AE, ABCDE
  - BC, BD, BE, BCDE
  - CD, CE, DE, CDE

Potential max-patterns

- Since BCDE is a max-pattern, no need to check BCD, BDE, CDE in later scan

Frequent Closed Patterns

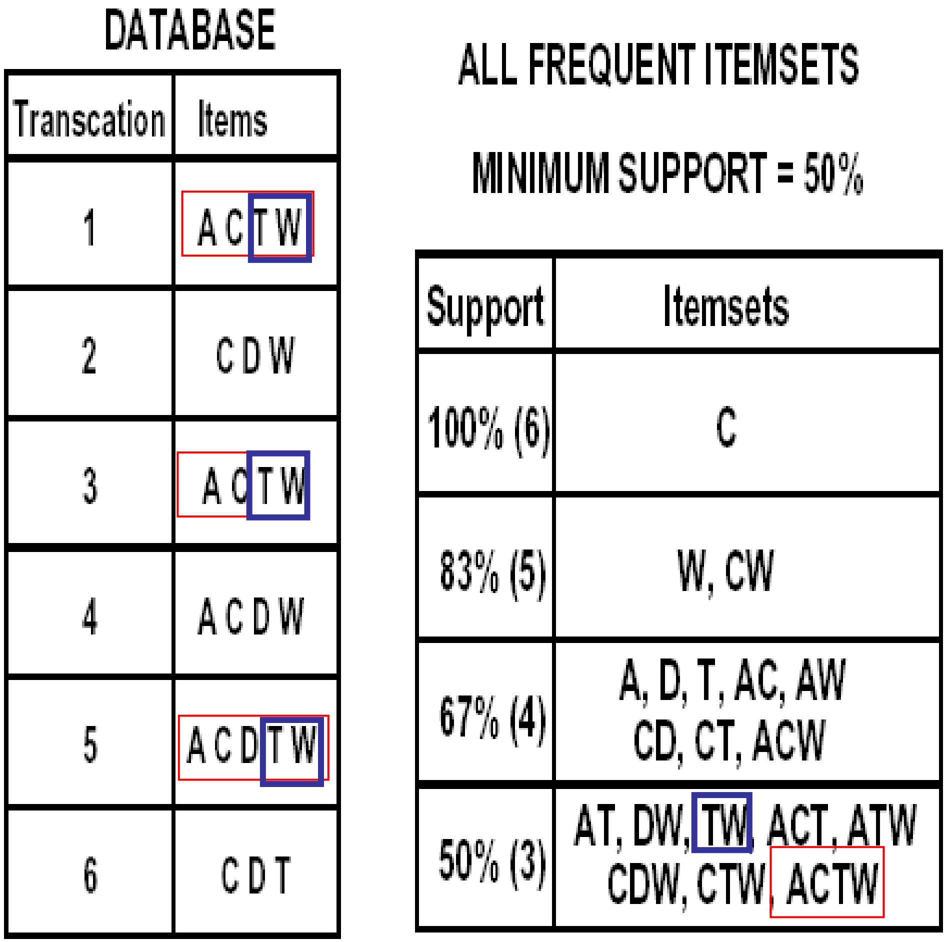
- For frequent itemset X, if there exists no item y s.t. every transaction containing X also contains y, then X is a frequent closed pattern
  - "acdf" is a frequent closed pattern
  - "ac" is not a frequent closed pattern
- Concise rep. of freq pats
- Reduce # of patterns and rules
- N. Pasquier et al. In ICDT'99

Min_sup=2

TID	Items
10	a, c, d, e, f
20	a, b, e
30	c, e, f
40	a, c, d, f
50	c, e, f

Frequent Closed Patterns(II)

- Another example:



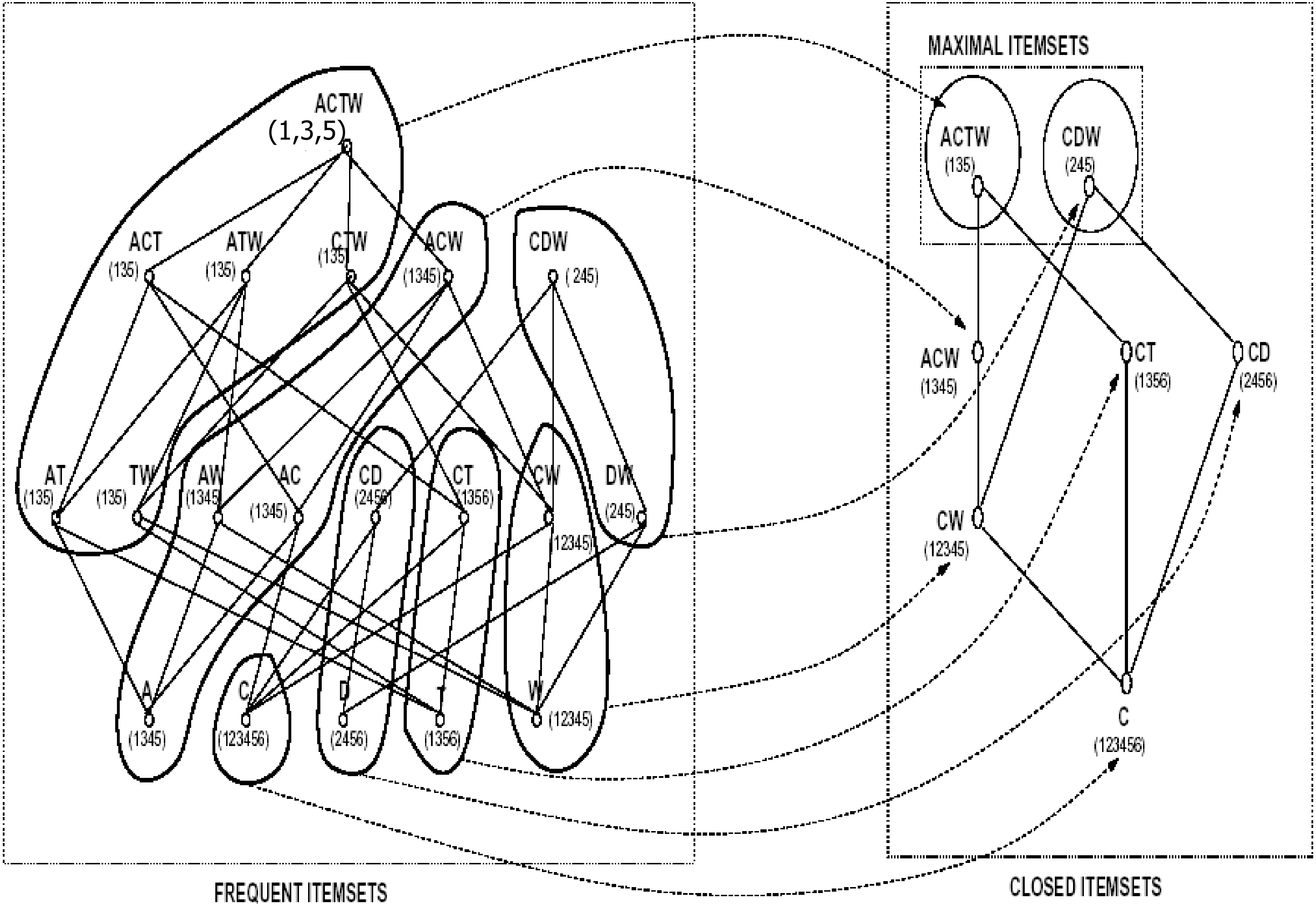
DATABASE

Transcation	Items
1	A C T W
2	C D W
3	A C T W
4	A C D W
5	A C D T W
6	C D T

ALL FREQUENT ITEMSETS

MINIMUM SUPPORT = 50%

Support	Itemsets
100% (6)	C
83% (5)	W, CW
67% (4)	A, D, T, AC, AW CD, CT, ACW
50% (3)	AT, DW, TW, ACT, ATW CDW, CTW, ACTW


MAXIMAL ITEMSETS
ACTW
(1,3,5)
ACT
(135)
ATW
(135)
CTW
(135)
ACW
(1345)
CDW
(245)
AT
(135)
TW
(135)
AW
(1345)
AC
(1345)
CD
(2456)
CT
(1356)
CW
(12345)
DW
(245)
A
(1345)
C
(123456)
D
(2456)
T
(1356)
W
(12345)
FREQUENT ITEMSETS
ACTW
(135)
CDW
(245)
ACW
(1345)
CT
(1356)
CD
(2456)
CW
(12345)
C
(123456)
CLOSED ITEMSETS

The CHARM Method

- Use vertical data format: t(AB)={T1, T12, …}
- Derive closed pattern based on vertical intersections
  - t(X)=t(Y): X and Y always happen together
  - $t(X) \subset t(Y)$: transaction having X always has Y
- Use diffset to accelerate mining
  - Only keep track of difference of tids
  - t(X)={T1, T2, T3}, $t(X_y)$={T1, T3}
  - Diffset(X_y, X)={T2}

CLOSET: Mining Frequent Closed Patterns

- Flist: list of all freq items in support asc. order
  - Flist: d-a-f-e-c
- Divide search space
  - Patterns having d
  - Patterns having d but no a, etc.
- Find frequent closed pattern recursively
  - Every transaction having d also has cfa → cfad is a frequent closed pattern
- PHM'00

Min_sup=2

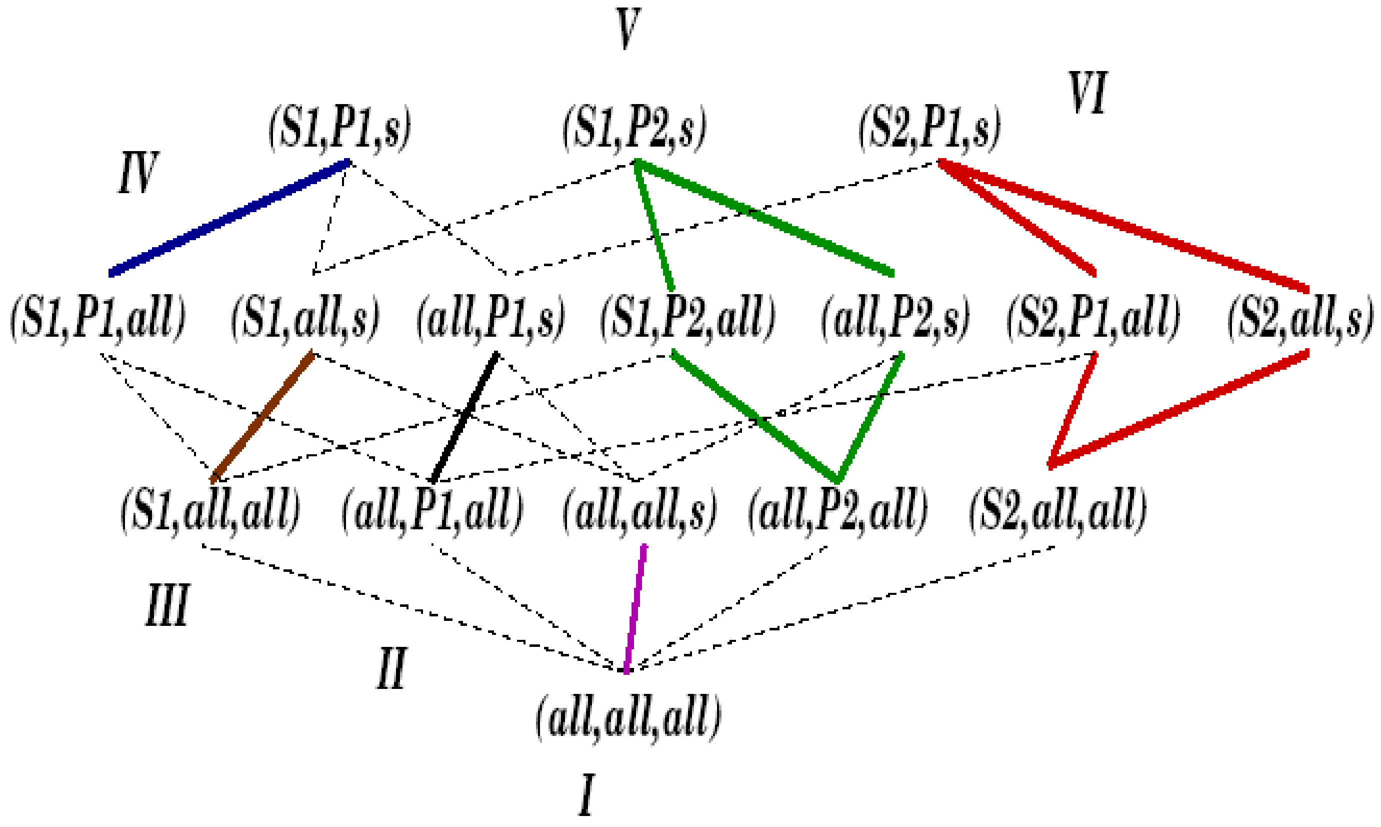
TID	Items
10	a, c, d, e, f
20	a, b, e
30	c, e, f
40	a, c, d, f
50	c, e, f

Closed and Max-patterns

- Closed pattern mining algorithms can be adapted to mine max-patterns
    - A max-pattern must be closed
- Depth-first search methods have advantages over breadth-first search ones

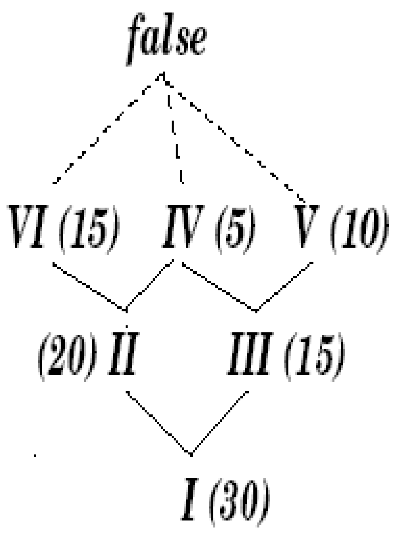
Generalizing to Closed Cube

- Laks V. S. Lakshmanan, Jian Pei, Jiawei Han: Quotient Cube: How to Summarize the Semantics of a Data Cube. VLDB 2002



T	P	S	M
S1	P1	s	5
S1	P2	s	10
S2	P1	s	15

(a)

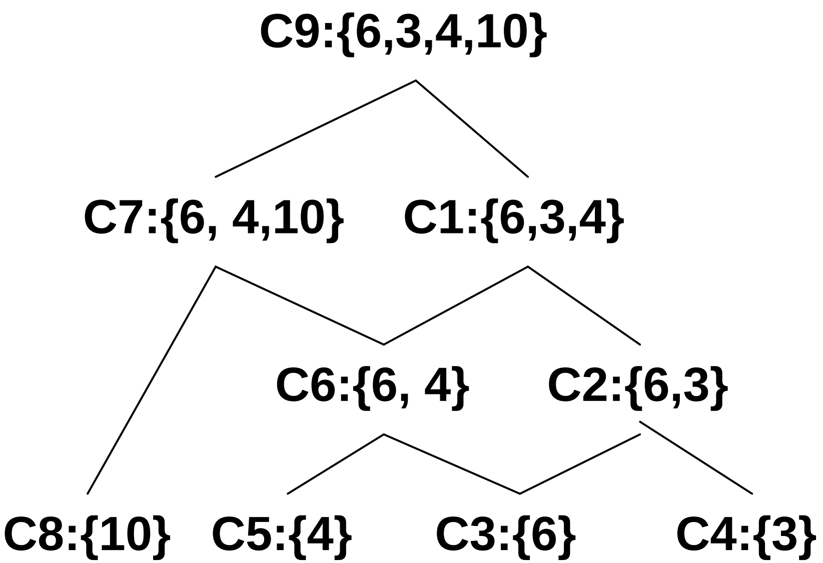


Maintaining Median in a Closed Cube

- What happen if we are storing the median?
- Maintain a data cube for holistic aggregation is hard
  - History tuple values must be kept in order to compute the new aggregate when tuples are inserted or deleted
- Maintain a closed cube with holistic aggregation makes the problem harder
  - Need to maintain the equivalence classes
- Use the concept of addset and sliding window
  - Cuiping Li, Gao Cong, Anthony K.H. Tung, Shan Wang. "Incremental Maintenance of Quotient Cube for Median". In Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'04) 2004

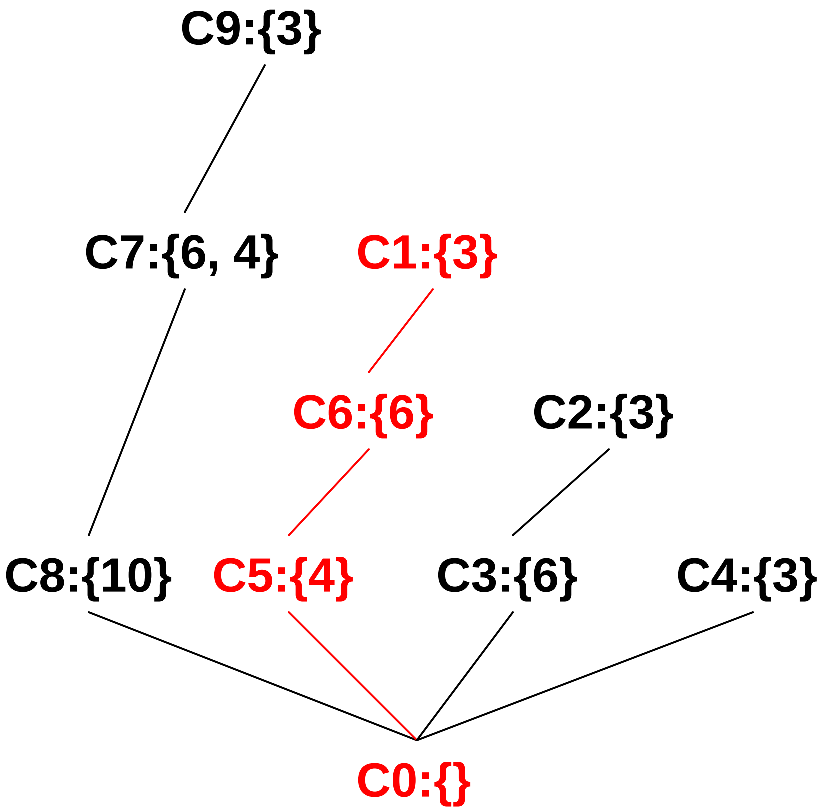
Maintaining Median in a Closed Cube(II)

- Closed cube helps to reduce this storage requirement
  - Store only one measure set for each equivalence class
- Problem: Redundancy! For example: 6,4,10 in C7 and C9



Naïve Addset data structure

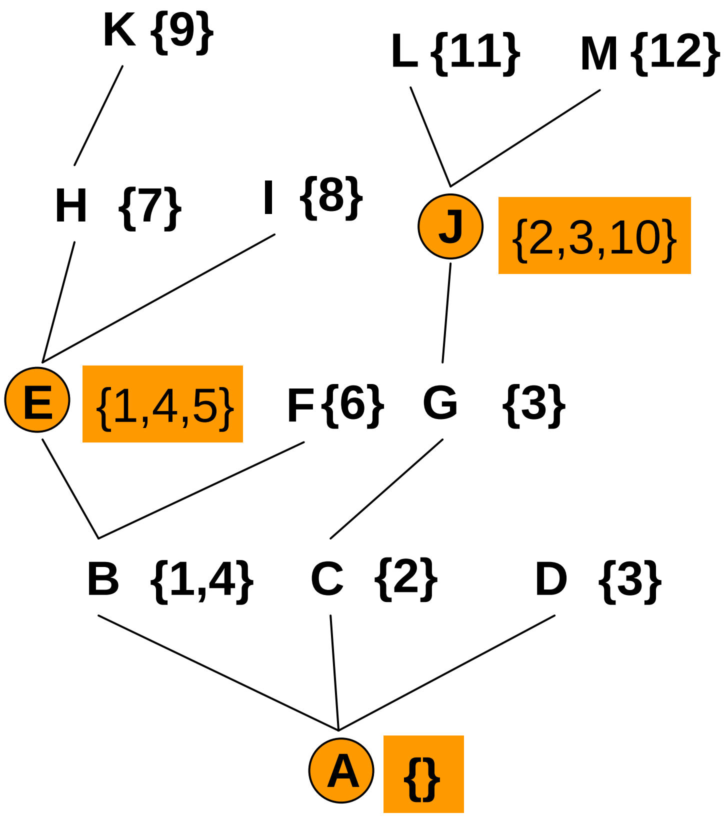
- Store the measure set difference between new class and its generator (Addset)
- Space saving: about 2 times (from 18 to 10)
- Family tree
- Combine the addset along its family linkage path, we get the measure set, for example: MS(C1)={3}∪{6}∪{4}={3,6,4}



Problem: Measure set computation cost increases with the length of family path

Dynamical Materialization of Addset

- In order to get some tradeoff between space and time, dynamically materialize some class in the process of maintenance
  - Pseudo class: store addset
  - Materialized class: store actual measure set
- To obtain the actual measure set of a pseudo class
  - only need to trace to its nearest materialized ancestor class instead of tree root.



Extension

- Extend the techniques to handle other holistic aggregations like quantile
- Approximation update
- Data stream
- …

Constrained Mining vs. Constraint-Based Search

- Constrained mining vs. constraint-based search
    - Both aim at reducing search space
    - Finding all patterns vs. some (or one) answers satisfying constraints
    - Constraint-pushing vs. heuristic search
    - An interesting research problem on integrating both
- Constrained mining vs. DBMS query processing
    - Database query processing requires to find all
    - Constrained pattern mining shares a similar philosophy as pushing selections deeply in query processing

Query Optimization for Constrained FPMA

- Mining frequent patterns with constraint C
  - Sound: only find patterns satisfying the constraints C
  - Complete: find all patterns satisfying the constraints C
- A naïve solution
  - Constraint test as a post-processing
- More efficient approaches
  - Analyze the properties of constraints comprehensively
  - Push constraints as deeply as possible inside the frequent pattern mining

Anti-Monotonicity

- Anti-monotonicity
  - An itemset S violates the constraint, so does any of its superset
  - sum(S.Profit) ≤ v is anti-monotone
  - sum(S.Profit) ≥ v is not anti-monotone
- Example
  - C: sum(S.Profit) ≥ 40 is anti-monotone
  - Itemset ac violates C
  - So does every superset of ac

TDB (min_sup=2)

TID	Transaction
10	a, b, c, d, f
20	b, c, d, f, g, h
30	a, c, d, e, f
40	c, e, f, g

Item	Profit
a	40
b	0
c	20
d	10
e	30
f	30
g	20
h	10

Monotonicity

TDB (min_sup=2)

TID	Transaction
10	a, b, c, d, f
20	b, c, d, f, g, h
30	a, c, d, e, f
40	c, e, f, g

- Monotonicity
  - An itemset S satisfies the constraint, so does any of its superset
  - sum(S.Price) ≥ v is monotone
  - min(S.Price) ≤ v is monotone
- Example
  - C: min(S.profit) ≤ 15
  - Itemset ab satisfies C
  - So does every superset of ab

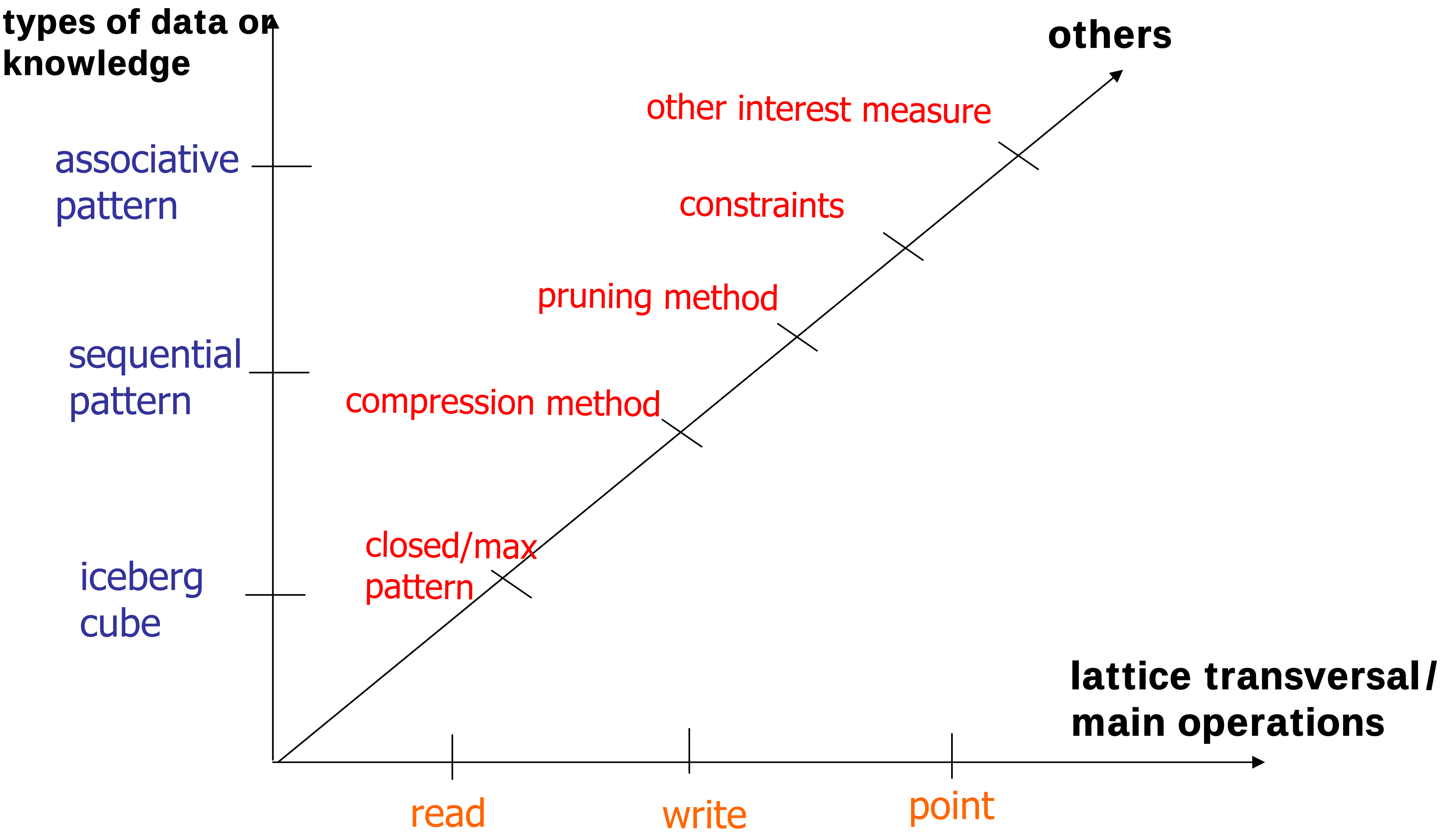
Item	Profit
a	40
b	0
c	20
d	10
e	30
f	30
g	20
h	10

Succinctness

- Succinctness:
  - Without even generating the itemset S, whether an itemset S satisfies constraint C can be determined based on the selection of items
  - min(S.Price) ≤ v is succinct
  - sum(S.Price) ≥ v is not succinct
- Example: min(S.Price)<20. We immediate know whether the constraint is satisfied once we generate the 2-itemset
- Optimization: If C is succinct, C is pre-counting pushable

Item	Profit
a	40
b	0
c	20
d	10
e	30
f	30
g	20
h	10

Summary


types of data or knowledge
associative pattern
sequential pattern
iceberg cube
others
other interest measure
constraints
pruning method
compression method
closed/max pattern
lattice transversal / main operations
read
write
point

Essential Readings

- http://www.cs.helsinki.fi/u/salmenki/dmcourse/chap12.pdf Chapter 2
- Mohammed J. Zaki, Srinivasan Parthasarathy, Mitsunori Ogihara, Wei Li, "**New Algorithms for Fast Discovery of Association Rules**", *3rd International Conference on Knowledge Discovery and Data Mining (KDD)*, pp 283-286, Newport, California, August, 1997. (Journal Version in 2000) **Note: The first paper the introduce depth first search for frequent pattern mining.**
- P.Shenoy, J. Haritsa, S. Sudarshan, G. Bhalotia, M. Bawa, D, Turbo-Charging Vertical Mining of Large Database." Proc. 2000 ACM-SIGMOD Int. Conf. on Management of Data (SIGMOD'00), Dallas, TX, May 2000. **Note: Tested on a dataset of 1.2GB, the largest that I have seen so far in data mining literatures. (Code Available Here)**
- Gao Cong, Beng Chin Ooi, Kian-Lee Tan, Anthony K. H. Tung, "Go Green: Recycle and Reuse Frequent Patterns". International Conference on Data Engineering (ICDE'2004), Boston, 2004
- [HK06] : "Data Mining: Concepts and Techniques", Chapter 5.1, 5.2, 5.4

References

- [AgSr94] R. Agrawal, R. Srikant, "**Fast Algorithms for Mining Association Rules**", Proc. of the 20th Int'l Conference on Very Large Databases, Santiago, Chile, Sept. 1994.
- [AgSr96] R. Srikant, R. Agrawal: "Mining Sequential Patterns: Generalizations and Performance Improvements", to appear in *Proc. of the Fifth Int'l Conference on Extending Database Technulogy (EDBT)*, Avignon, France, March 1996.
- *[BeRa99]* Kevin S. Beyer and Raghu Ramakrishnan. "***Bottom-Up Computation of Sparse and Iceberg CUBEs***". In Proceedings of the ACM SIGMOD International Conference on Management of Data, pages 359-370, June 1999.
- [HPDW01] J. Han, J. Pei, G. Dong, and K. Wang, "**Efficient Computation of Iceberg Cubes with Complex Measures**", Proc. 2001 ACM-SIGMOD Int. Conf. on Management of Data (SIGMOD'01), Santa Barbara, CA, May 2001.
- [HPY00] J. Han, J. Pei, and Y. Yin, `` **Mining Frequent Patterns without Candidate Generation''**,, Proc. 2000 ACM-SIGMOD Int. Conf. on Management of Data (SIGMOD'00), Dallas, TX, May 2000.
- [HaPe00] J. Han and J. Pei ``**Mining Frequent Patterns by Pattern-Growth: Methodology and Implications** '', ACM SIGKDD Explorations (Special Issue on Scaleble Data Mining Algorithms), 2(2), December 2000.
- [PHPC01] J. Pei, J. Han, H. Pinto, Q. Chen, U. Dayal, and M.-C. Hsu, " **PrefixSpan: Mining Sequential Patterns Efficiently by Prefix-Projected Pattern Growth''**, Proc. 2001 Int. Conf. on Data Engineering (ICDE'01), Heidelberg, Germany, April 2001.

References

- [Pei01] J. Pei, J. Han, H. Pinto, Q. Chen, U. Dayal, and M.-C. Hsu, " *PrefixSpan: Mining Sequential Patterns Efficiently by Prefix-Projected Pattern Growth* ", *Proc. 2001 Int. Conf. on Data Engineering (ICDE'01), Heidelberg, Germany, April 2001.*
- [SHSB00] P.Shenoy, J. Haritsa, S. Sudarshan, G. Bhalotia, M. Bawa, D. "**Turbo-Charging Vertical Mining of Large Database.**" Proc. 2000 ACM-SIGMOD Int. Conf. on Management of Data (SIGMOD'00), Dallas, TX, May 2000.
- [Zaki97]Mohammed J. Zaki, Srinivasan Parthasarathy, Mitsunori Ogihara, Wei Li, "New Algorithms for Fast Discovery of Association Rules", 3rd International Conference on Knowledge Discovery and Data Mining (KDD), pp 283-286, Newport, California, August, 1997
- [Zaki98] Mohammed J. Zaki, "**Efficient Enumeration of Frequent Sequences**", 7th International Conference on Information and Knowledge Management, pp 68-75, Washington DC, November 1998.
- [Zaki00] Mohammed J. Zaki, "**Scalable Algorithms for Association Mining**", in IEEE Transactions on Knowledge and Data Engineering, Vol. 12, No. 3, pp 372-390, May/June 2000
- [Zaki01] Mohammed J. Zaki, "**SPADE: An Efficient Algorithm for Mining Frequent Sequences**", in Machine Learning Journal, special issue on Unsupervised Learning (Doug Fisher, ed.), pp 31-60, Vol. 42 Nos. 1/2, Jan/Feb 2001