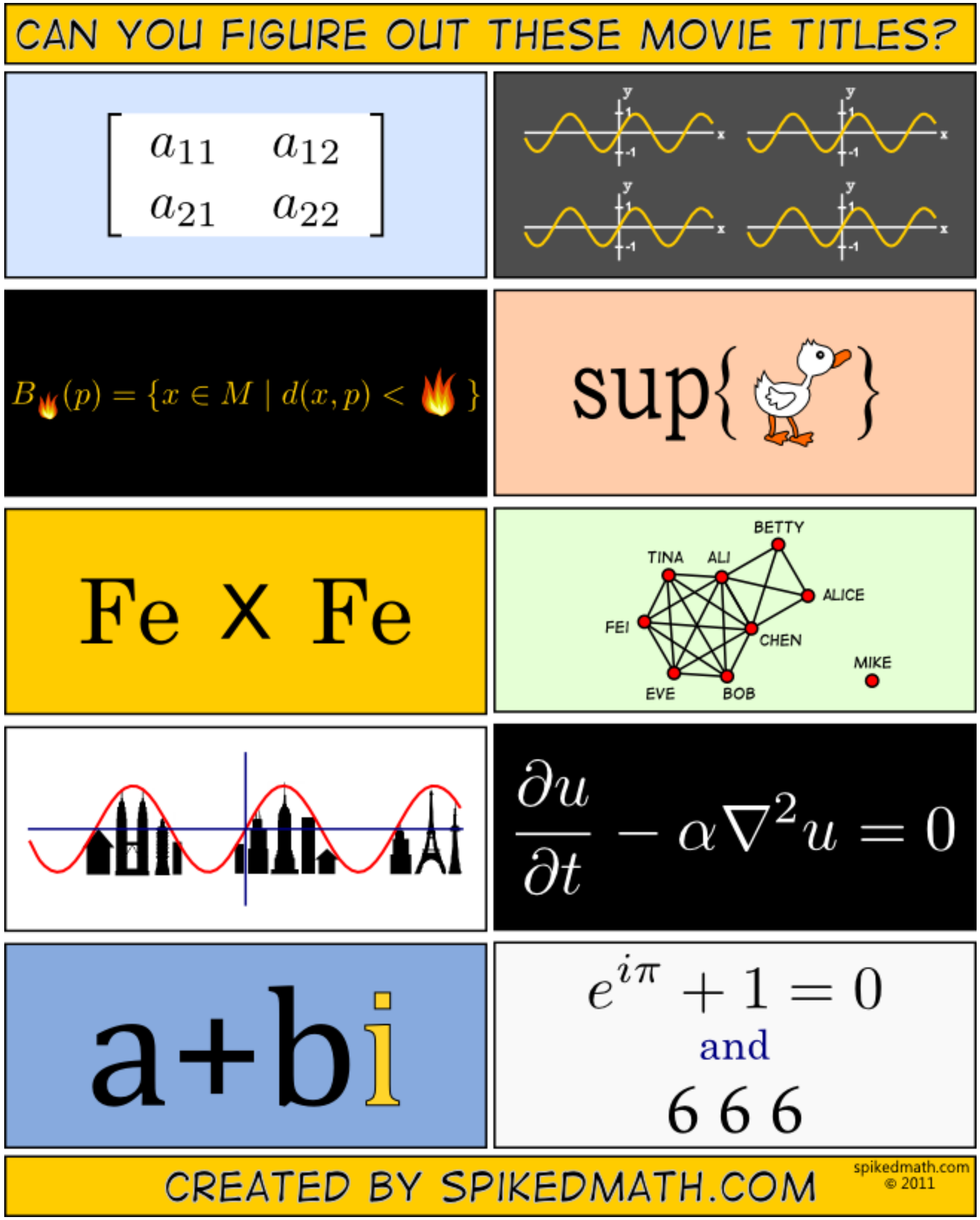

CAN YOU FIGURE OUT THESE MOVIE TITLES?
$\begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix}$
sup{ }
Fe X Fe
BETTY
TINA
ALI
ALICE
FEI
CHEN
MIKE
EVE
BOB
$\frac{\partial u}{\partial t} - \alpha \nabla^2 u = 0$
a+bi
$e^{i\pi} + 1 = 0$
and
6 6 6
CREATED BY SPIKEDMATH.COM
spikedmath.com
© 2011

Data Mining: Association Analysis

Laura Brown

Some slides adapted from G. Piatetsky-Shapiro; Han, Kamber, & Pei; Tan, Steinbach, & Kumar

Association Rule Mining

- Given a set of transaction, find rules that will predict the occurrence of an item based on the occurrences of other items in the transaction

Market-Basket transactions

TID	*Items*
1	**Bread, Milk**
2	**Bread, Diaper, Beer, Eggs**
3	**Milk, Diaper, Beer, Coke**
4	**Bread, Milk, Diaper, Beer**
5	**Bread, Milk, Diaper, Coke**

Example of Association Rules

{Diaper} -> {Beer},
{Milk, Bread} -> {Eggs,Coke},
{Beer, Bread} -> {Milk},

Implication means co-occurrence, not causality!

Applications

- Example 1 – text mining
  - baskets = sentences
  - items = words in those sentences
    - find words that appear together unusually frequently, i.e., linked concepts
- Example 2 – document mining
  - baskets = sentences
  - items = documents containing those sentences
    - items that appear together too often could represent plagiarism

Applications

- Example 3 – healthcare mining
  - baskets = people
  - items = genes or blood-chemistry factors
    - detect combinations of genes that results in a disease
    - requires extension: absence of an item needs to be observed as well as presence

Terminology

Association Analysis: Frequent itemset mining

Terminology

- Itemsets – a set of items (collection of one or more items, $X \subseteq I$
    - *Items* : $I = \{x_1, x_2, \ldots, x_m\}$
    - *k-itemset* – an itemset with *k* items
    - Ex. *X* = {*Milk, Bread, Beer*}

- Tidsets – a set of *tids,* $T \subseteq \mathcal{T}$
    - *Transaction identifiers* or *tids* : $\mathcal{T} = \{t_1, t_2, \ldots, t_n\}$

- Transaction – a transaction is a tuple of the form $\langle t, X \rangle$, where $t \in \mathcal{T}$ is a unique *tid* and *X* is an itemset

Database

D	*A*	*B*	*C*	*D*	*E*
1	1	1	0	1	1
2	0	1	1	0	1
3	1	1	0	1	1
4	1	1	1	0	1
5	1	1	1	1	1
6	0	1	1	1	0

Binary Database

t	$\mathbf{i}(t)$
1	*ABDE*
2	*BCE*
3	*ABDE*
4	*ABCE*
5	*ABCDE*
6	*BCD*

Transaction Database

$\mathbf{t}(x)$				
A	*B*	*C*	*D*	*E*
1	1	2	1	1
3	2	4	3	2
4	3	5	5	3
5	4	6	6	4
	5			5
	6			

Vertical Database

- The database **D** has 5 items, $I = \{A, B, C, D, E\}$ and 6 *tids* $\mathcal{T} = \{1, 2, 3, 4, 5, 6\}$
- When describing a transaction we can drop the set notation: $\langle 1, \{A, B, D, E\}\rangle \rightarrow \langle t, ABDE\rangle$

Support and Frequent Itemsets

- The support of an itemset X, $sup(X)$ or $\sigma(X)$, in **D** is the number of transactions in **D** that contain X
  - Ex. $sup(\{\text{Milk, Bread, Diaper}\}) = 2$

TID	*Items*
1	**Bread, Milk**
2	**Bread, Diaper, Beer, Eggs**
3	**Milk, Diaper, Beer, Coke**
4	**Bread, Milk, Diaper, Beer**
5	**Bread, Milk, Diaper, Coke**

- The relative support of an itemset X, $rsup(X)$ or $s(X)$, is the fraction of transactions that contain X

$$rsup(X) = \frac{\mathrm{sup}(X)}{|\boldsymbol{D}|},$$ Ex. $rsup(\{\text{Milk, Bread, Diaper}\}) = 2 / 5$

- An itemset X is frequent in **D** if $sup(X) \geq minsup$, where *minsup* is a user defined *minimum support threshold*

Frequent Itemsets

minsup = 3

t	$\mathbf{i}(t)$
1	*ABDE*
2	*BCE*
3	*ABDE*
4	*ABCE*
5	*ABCDE*
6	*BCD*

Transaction Database

sup	itemsets
6	*B*
5	*E*, *BE*
4	*A*, *C*, *D*, *AB*, *AE*, *BC*, *BD*, *ABE*
3	*AD*, *CE*, *DE*, *ABD*, *ADE*, *BCE*, *BDE*, *ABDE*

Frequent Itemsets

- The set $\mathcal{F}$ is the set of all frequent itemsets, and $\mathcal{F}^{(k)}$ is the set of frequent k-itemsets
- The 19 frequent itemsets comprise the set $\mathcal{F}$

$$\mathcal{F}^{(1)} = \{A, B, C, D, E\}$$
$$\mathcal{F}^{(2)} = \{AB, AD, AE, BC, BD, BE, CE, DE\}$$
$$\mathcal{F}^{(3)} = \{ABD, ABE, ADE, BCE, BDE\}$$
$$\mathcal{F}^{(4)} = \{ABDE\}$$

Example: Frequent Itemsets

- Items = { milk, coke, pepsi, beer, juice }
- MinSupport = 3 baskets

t_1 = {m, c, b} t_2 = {m, p, j}
t_3 = {m, b} t_4 = {c, j}
t_5 = {m, p, b} t_6 = {m, c, b, j}
t_7 = {c, b, j} t_8 = {b, c}

- Frequent itemsets:

Example: Frequent Itemsets

- Items = { milk, coke, pepsi, beer, juice }
- MinSupport = 3 baskets

t_1 = {m, c, b}
t_2 = {m, p, j}
t_3 = {m, b}
t_4 = {c, j}
t_5 = {m, p, b}
t_6 = {m, c, b, j}
t_7 = {c, b, j}
t_8 = {b, c}

- Frequent itemsets:
  - {m}, {c}, {b}, {j}, {m, b}, {b, c}, {c, j}
  - $\mathcal{F}^{(1)}$ = {m, c, b, j}
  - $\mathcal{F}^{(2)}$ = {mb, bc, cj}

Association Rules

- An association rule is an expression of the form

  $$X \text{ -> } Y$$

  where X and Y are disjoint itemsets. Denote $X \cup Y$ as XY

  Ex. {Milk, Diaper} -> {Beer}

- The support of a rule is the number of transactions in which X and Y co-occur

  $$s = sup(X \text{ -> } Y) = sup(XY)$$

- The relative support of a rule is the fraction of transactions in which X and Y co-occur

  $$rsup(X \text{ -> } Y) = sup(XY) / |\mathbf{D}| = \mathrm{P}(X \wedge Y)$$

- The confidence of a rule is the conditional probability that a transaction contains Y given that it contains X

  $$c = conf(X \text{ -> } Y) = \mathrm{P}(Y \mid X) = \mathrm{P}(X \wedge Y) / \mathrm{P}(X) = sup(\mathrm{XY}) / sup(X)$$

Frequent Itemset Mining

Association Analysis

Mining Association Rules

Two-step approach

1. Frequent Itemset Generation

   generate all itemsets whose support ≥ *minsup*

2. Rule Generation

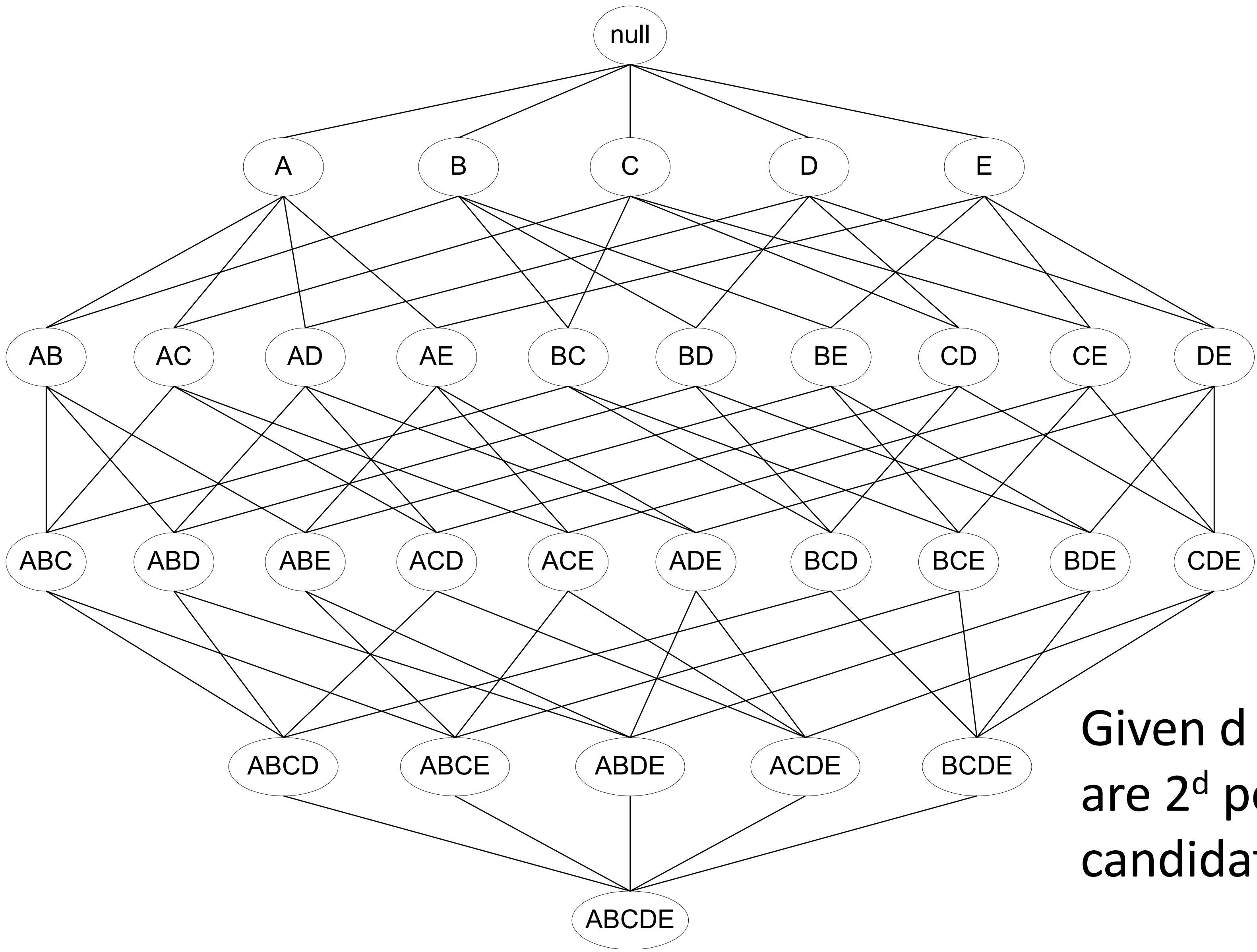
   generate high confidence rules from each frequent itemset, where each rule is a binary partitioning of a frequent itemset

Frequent Itemset Generation

- Frequent itemset generation is computationally expensive

- How many itemsets are potentially to be generated in the worst case?
  - number is sensitive to the *minsup* threshold
  - when *minsup* is low, there exists potentially an exponential number of frequent itemsets

Frequent Itemset Generation



Given d items, there are 2^d possible candidate itemsets

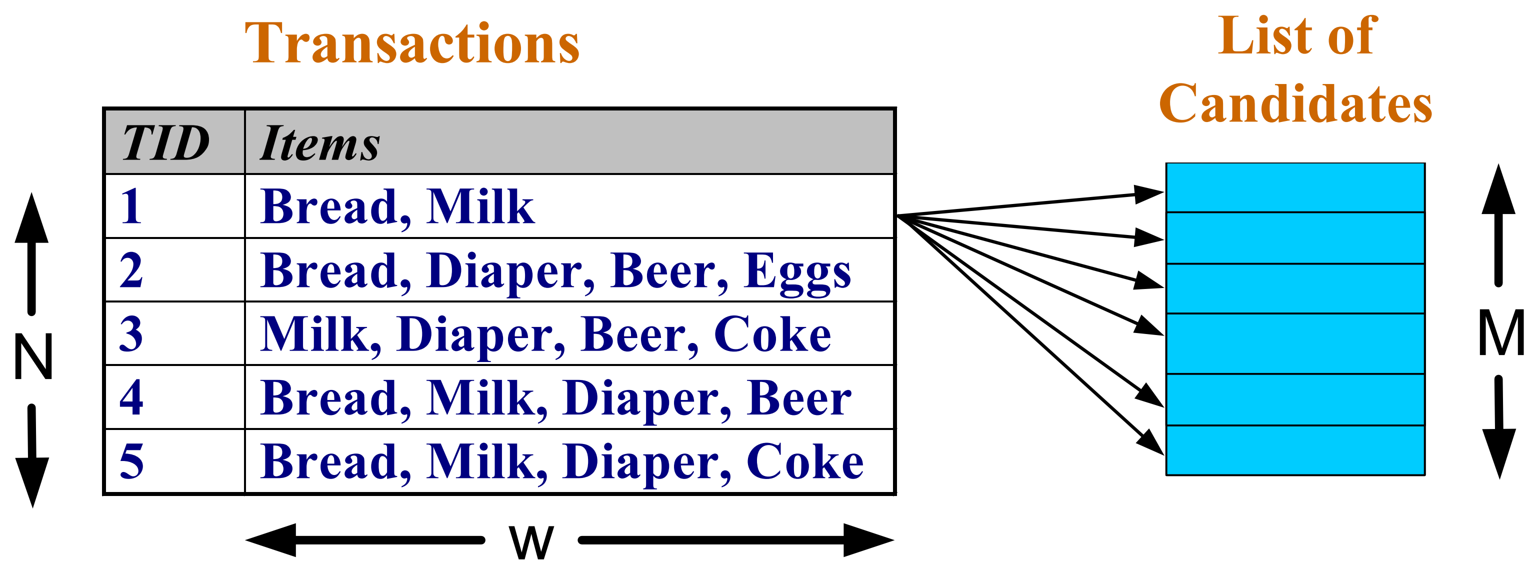
Subset Property

- Every subset of a frequent set is frequent!
  - If {A, B} is frequent. Each occurrence of A, B includes both A and B, then both A and B alone must also be frequent
- A long pattern (itemsets) contains a combinatorial number of sub-patterns (itemsets)
  - A frequent set with 100 items contains

$$\binom{100}{1} + \binom{100}{2} + \cdots + \binom{100}{100} = 2^{100} - 1$$

Brute Force Algorithm

- Each itemset in the lattice is a candidate frequent itemset
- Count the support of each candidate by scanning the database



TID	*Items*
1	Bread, Milk
2	Bread, Diaper, Beer, Eggs
3	Milk, Diaper, Beer, Coke
4	Bread, Milk, Diaper, Beer
5	Bread, Milk, Diaper, Coke

- Match each transaction against every candidate
- Complexity ~O(NMw) -> expensive M = 2^d

Frequent Itemset Generation Strategies

- Reduce the number of candidates (M)
  - complete search: $M=2^d$
  - use pruning methods to reduce M
- Reduce the number of transactions (N)
  - reduce size of N as the size of itemset increases
  - used by DHP and vertical-based mining algorithms
- Reduce the number of comparisons (NM)
  - use efficient data structures to store candidates or transactions
  - no need to match every candidate against every transaction

Apriori Frequent Itemset Mining

Association Analysis

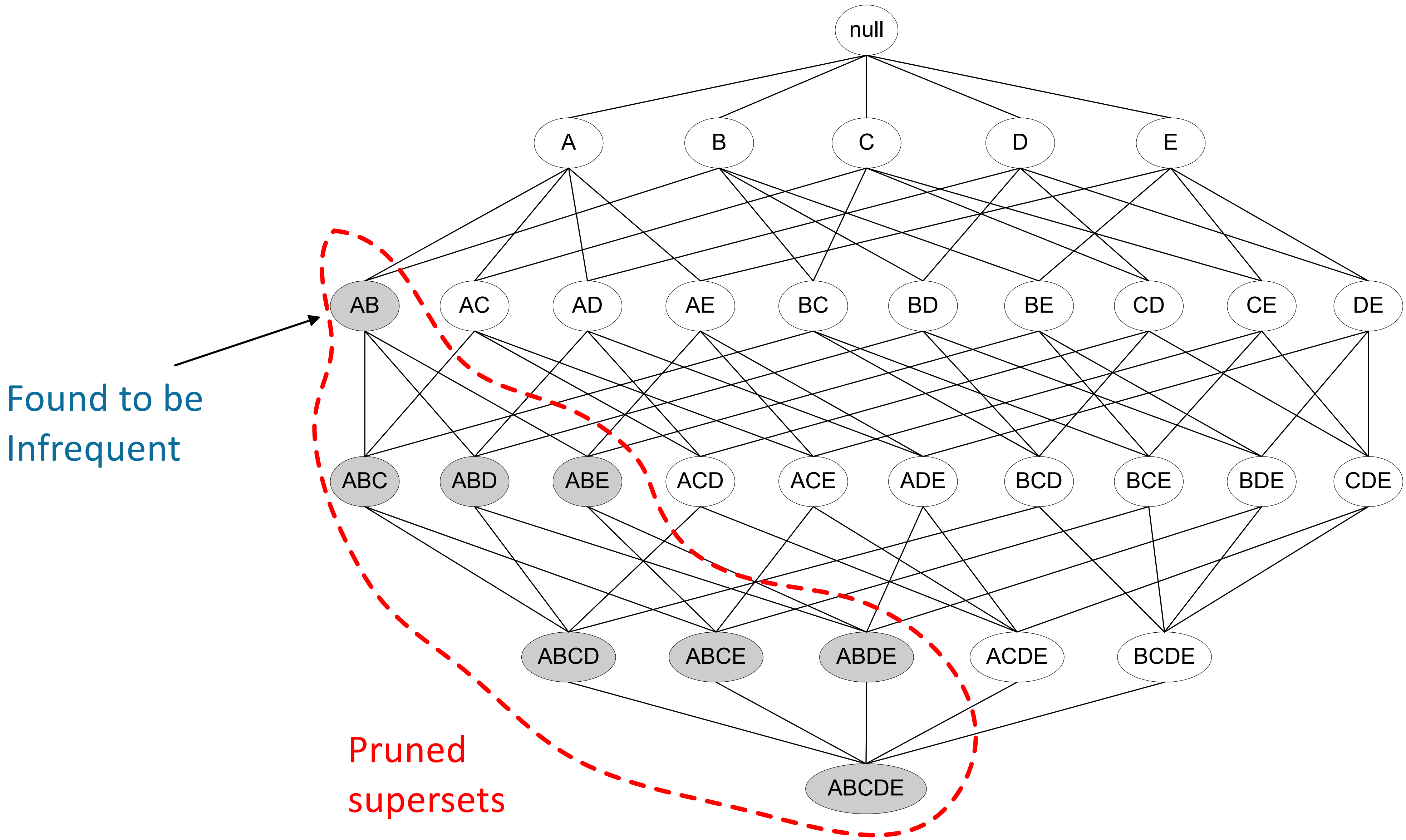
Apriori

- Apriori principle:
  1. If an itemset *X* is frequent, then all of its subsets *Y*, $Y \subseteq X$ must also be frequent
  2. If an itemset *X* is not frequent, its supersets *Y*, $Y \supseteq X$ are also not frequent
- Apriori uses these principles to improve the efficiency of the search for frequent itemsets

- Anti-monotone property of support
  - support of an itemset never exceeds the support of its subsets

$$\forall X, Y : (X \subseteq Y) \Rightarrow sup(X) \geq sup(Y)$$

Illustrating Apriori Principle



Illustrating Apriori Principle

Item	Count
Bread	**4**
Coke	**2**
Milk	**4**
Beer	**3**
Diaper	**4**
Eggs	**1**

Items (1-itemsets)

Itemset	Count
{Bread,Milk}	**3**
{Bread,Beer}	**2**
{Bread,Diaper}	**3**
{Milk,Beer}	**2**
{Milk,Diaper}	**3**
{Beer,Diaper}	**3**

Pairs (2-itemsets)

(No need to generate candidates involving Coke or Eggs)

Minimum Support = 3

Triplets (3-itemsets)

Itemset	Count
{Bread,Milk,Diaper}	**3**

If every subset is considered:

$C_{6,1} + C_{6,2} + C_{6,3} = 41$

With support-based pruning

$6 + 6 + 1 = 13$

Apriori Pseudocode

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

C_1 = scan database to find support for each item;
L_1 = {frequent items};
for (k = 1; L_k !=emptyset; 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 $\cup_k L_k$;

Apriori Algorithm - Example

$Sup_{min} = 2$

Database TDB

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

1st scan

C_1

Itemset	sup
{A}	2
{B}	3
{C}	3
{D}	1
{E}	3

L_1

Itemset	sup
{A}	2
{B}	3
{C}	3
{E}	3

C_2

Itemset
{A, B}
{A, C}
{A, E}
{B, C}
{B, E}
{C, E}

2nd scan

C_2

Itemset	sup
{A, B}	1
{A, C}	2
{A, E}	1
{B, C}	2
{B, E}	3
{C, E}	2

L_2

Itemset	sup
{A, C}	2
{B, C}	2
{B, E}	3
{C, E}	2

C_3

Itemset
{B, C, E}

3rd scan

L_3

Itemset	sup
{B, C, E}	2

Factors Affecting Complexity

- Choice of minimum support threshold
  - lowering support threshold results in more frequent itemsets
  - this may increase number of candidates and max length of frequent itemsets
- Dimensionality (number of items) of the data set
  - more space is needed to store support count of each item
  - if number of frequent items also increases, both computation and I/O costs may also increase
- Size of database
  - since Apriori makes multiple passes, run time of algorithm may increase with number of transactions
- Average transaction width
  - transaction width increases with denser data sets
  - This may increase max length of frequent itemsets and traversals of hash tree (number of subsets in a transaction increases with its width)

Improvements to Apriori

- Partition DB, find local frequent patterns, consolidate to global patterns
  - Savasere, Omiecinski, and Navathe, VLDB, 1995
- Reduce number of candidates with DHP
  - Park, Chen, and Yu, SIGMOD, 1995
- Sampling for frequent patterns, verify pattern in db
  - Toivonen, VLDB, 1996.
- Dynamic Itemset counting (DIC)
  - Brin, Motwani, Ullman, Tsur, SIGMOD, 1997

Apriori Example

Association Analysis

Apriori Algorithm Example

- Consider DB of 9 transactions
- *minsup* = 2

TID	Items
T100	I1, I2, I5
T101	I2, I4
T102	I2, I3
T103	I1, I2, I4
T104	I1, I3
T105	I2, I3
T106	I1, I3
T107	I1, I2, I3, I5
T108	I1, I2, I3

Step 1: Generate 1-itemset patterns

minsup = 2

Scan D for count of each candidate

Itemset	Sup.Count
{I1}	6
{I2}	7
{I3}	6
{I4}	2
{I5}	2

C_1

Compare candidate support count with minimum support count

Itemset	Sup.Count
{I1}	6
{I2}	7
{I3}	6
{I4}	2
{I5}	2

L_1

- In first iteration, each item is a member of the set of candidates
- Set of frequent 1-itemset, L_1, consists of candidate 1-itemsets satisfying minimum support

Step 2: Generate 2-itemset patterns

minsup = 2

Itemset	Sup.Count
{I1}	6
{I2}	7
{I3}	6
{I4}	2
{I5}	2

L_1

Generate C_2 candidates from L_1

Itemset
{I1, I2}
{I1, I3}
{I1, I4}
{I1, I5}
{I2, I3}
{I2, I4}
{I2, I5}
{I3, I4}
{I3, I5}
{I4, I5}

C_2

Scan D for count of each candidate

Itemset	Sup. Count
{I1, I2}	4
{I1, I3}	4
{I1, I4}	1
{I1, I5}	2
{I2, I3}	4
{I2, I4}	2
{I2, I5}	2
{I3, I4}	0
{I3, I5}	1
{I4, I5}	0

C_2

Compare candidate support count with minimum support count

Itemset	Sup Count
{I1, I2}	4
{I1, I3}	4
{I1, I5}	2
{I2, I3}	4
{I2, I4}	2
{I2, I5}	2

L_2

Step 3: Generate 3-itemset patterns

minsup = 2

Itemset	Sup Count
{I1, I2}	4
{I1, I3}	4
{I1, I5}	2
{I2, I3}	4
{I2, I4}	2
{I2, I5}	2

L_2

Generate C_3 from L_2

Itemset
{I1, I2, I3}
{I1, I2, I5}

C_3

Scan D for count of each candidate

Itemset	Sup. Count
{I1, I2, I3}	2
{I1, I2, I5}	2

C_3

Compare candidate support count with min support count

Itemset	Sup Count
{I1, I2, I3}	2
{I1, I2, I5}	2

L_3

- Generation of set of candidate 3-itemsets, C_3, involves use of Apriori property
- To find C_3, compute L_2 *Join* L_2
- C_3 = { {I1, I2, I3}, {I1 I2, I5}, {I1, I3, I5}, {I2, I3, I4}, {I2, I3, I5}, {I2, I4, I5} }
- Join step complete, prune step used to reduce size of C_3

Step 3: Generate 3-itemset patterns

- Use Apriori property: all subsets of frequent itemsets must also be frequent
- Ex. {I1, I2, I3} has 2-itemsets subsets of:
  - {I1, I2}, {I1, I3}, {I2, I3} are all in L_2,
  - keep {I1, I2, I3} in C_3
- Ex. {I2, I3, I5} has 2-itemsets subsets of:
  - {I2, I3}, {I2, I5}, {I3, I5}
  - {I3, I5} not a member of L_2, thus {I2, I3, I5} not in C_3
- Therefore, C_3 = {{I1, I2, I3}, {I1, I2, I5}}

Step 3: Generate 3-itemset patterns

minsup = 2

Itemset	Sup Count
{I1, I2}	4
{I1, I3}	4
{I1, I5}	2
{I2, I3}	4
{I2, I4}	2
{I2, I5}	2

L_2

Generate C_3 from L_2

Itemset
{I1, I2, I3}
{I1, I2, I5}

C_3

Scan D for count of each candidate

Itemset	Sup. Count
{I1, I2, I3}	2
{I1, I2, I5}	2

C_3

Compare candidate support count with min support count

Itemset	Sup Count
{I1, I2, I3}	2
{I1, I2, I5}	2

L_3

- Generation of set of candidate 3-itemsets, C_3, involves use of Apriori property
- Prune operation used to reduce size of C_3

Step 4: Generate 4-itemset patterns

- Algorithm uses L_3 *Join* L_3 to generate 4-itemsets, C_4.
  - Join results in { {I1, I2, I3, I5} }
  - itemset is pruned since { {I2, I3, I5} } is not frequent
- Algorithm terminates, having found all frequent items

$$\mathcal{F} = \{L_1, L_2, L_3\} = \{1, 2, 3, 4, 5, 12, 13, 15, 23, 24, 25, 123, 125\}$$

- Still left to do
  - generate association rules from itemsets
  - improve efficiency

Frequent Itemset Mining with Vertical Data

Association Analysis: ECLAT

Frequent Mining with Vertical Data

- Vertical format
  - for each item store a list of transaction IDs (tids)
- tid-list: list of tids for itemsets
  - $t(AB) = \{T_{11}, T_{25}, \ldots\}$
- Derive frequent patterns 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

ECLAT – Equivalence Class Transformation

- For each item, store a list of transaction ids (tids)

Horizontal Data Layout

TID	Items
1	A,B,E
2	B,C,D
3	C,E
4	A,C,D
5	A,B,C,D
6	A,E
7	A,B
8	A,B,C
9	A,C,D
10	B

Vertical Data Layout

A	B	C	D	E
1	1	2	2	1
4	2	3	4	3
5	5	4	5	6
6	7	8	9	
7	8	9		
8	10			
9				

TID-list

ECLAT

- Determine support of any k-itemset by intersection

A
1
4
5
6
7
8
9

$\cap$

B
1
2
5
7
8
10

=

AB
1
5
7
8

- Use diffset to accelerate mining
  - only keep track of difference of tids
  - Diffset(AB, A) = { 4, 6, 9 }, Diffset(AB, B) = { 2, 10 }

ECLAT

- 3 traversal approaches for itemsets
  - top-down, bottom-up, and hybrid
- Advantages: very fast support counting
- Disadvantages: intermediate tid-lists may become too large for memory

- References:
  - ECLAT – Zaki et al., KDD 1997
  - Mining closed patterns with vertical format: CHARM – Zaki & Hsiao, SDM 2002

Frequent Pattern Tree Methods: FPGrowth

Association Analysis: FPGrowth

Algorithms for Mining Frequent Patterns

- Bottlenecks of Apriori
  - breadth-first (i.e., level-wise) search
  - candidate generation and test
    - may generate huge number of candidates
- FPGrowth Approach (Han, Pei, Yin SIGMOD, 2000)
  - depth-first search
  - avoid explicit candidate generation
- Main Idea – grow long patterns from short ones using local frequent items only

Construct FP-tree

- Compress a large database into a compact, Frequent-Pattern tree (FP-tree) structure
  - highly condensed, but complete for frequent pattern mining
  - helps avoid costly database scans
- Develop an efficient, FP-tree based frequent pattern mining method
  - divide and conquer methodology: decompose mining tasks into smaller ones
  - avoid candidate generation: sub-database test only

Construct FP-tree: Overview

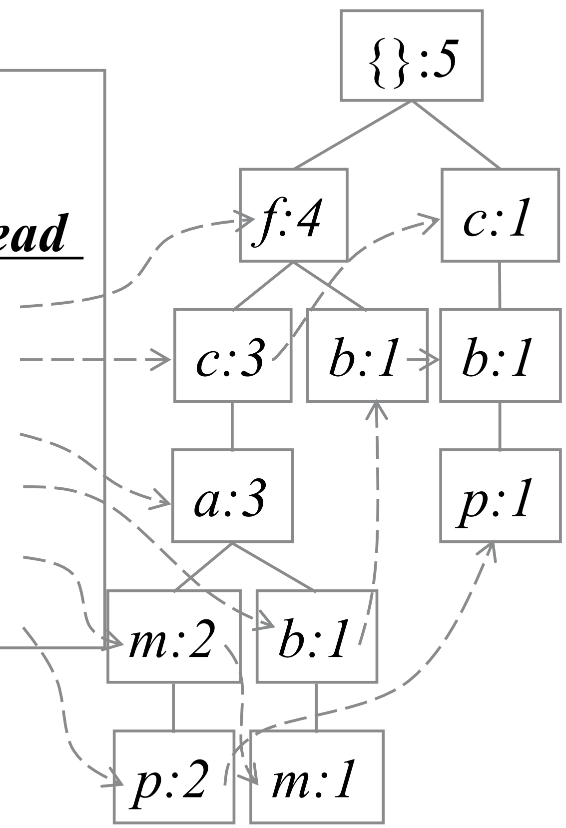
TID	*Items bought*	*(ordered) frequent items*
100	***{f, a, c, d, g, i, m, p}***	***{f, c, a, m, p}***
200	***{a, b, c, f, l, m, o}***	***{f, c, a, b, m}***
300	***{b, f, h, j, o, w}***	***{f, b}***
400	***{b, c, k, s, p}***	***{c, b, p}***
500	***{a, f, c, e, l, p, m, n}***	***{f, c, a, m, p}***

min_support = 3

1. Scan DB once, find frequent 1-itemset (single item pattern)
2. Sort frequent items in frequency descending order, f-list
3. Scan DB again, construct FP-tree

Header Table

Item	*frequency*	*head*
f	*4*	
c	*4*	
a	*3*	
b	*3*	
m	*3*	
p	*3*	



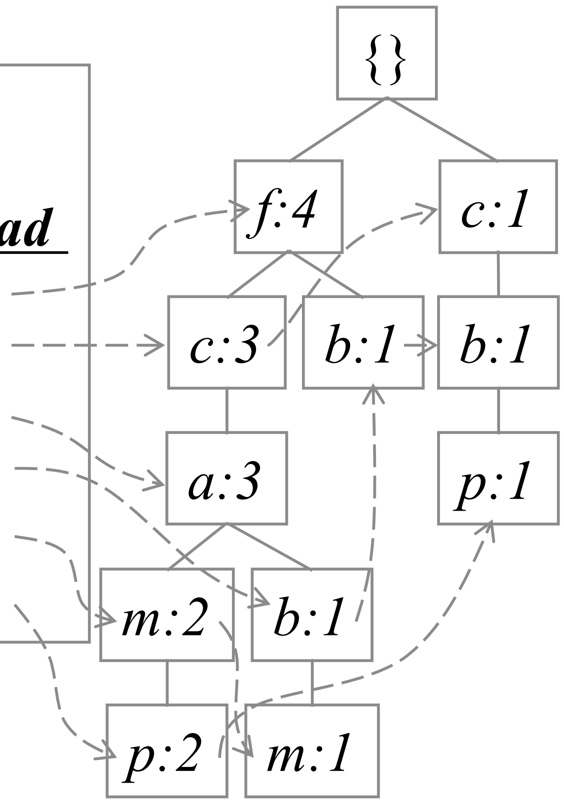
F-list = f-c-a-b-m-p

Find Patterns using FP-Tree

- Starting at the frequent item header table in the FP-tree
- Traverse the FP-tree by following the link of each frequent item *p*
- Accumulate all of the transformed prefix paths of item *p* to form *p*'s conditional pattern base

Header Table

Item	***frequency***	***head***
f	*4*	
c	*4*	
a	*3*	
b	*3*	
m	*3*	
p	*3*	



***Conditional* pattern bases**

item	***cond. pattern base***
c	***f:3***
a	***fc:3***
b	***fca:1, f:1, c:1***
m	***fca:2, fcab:1***
p	***fcam:2, cb:1***

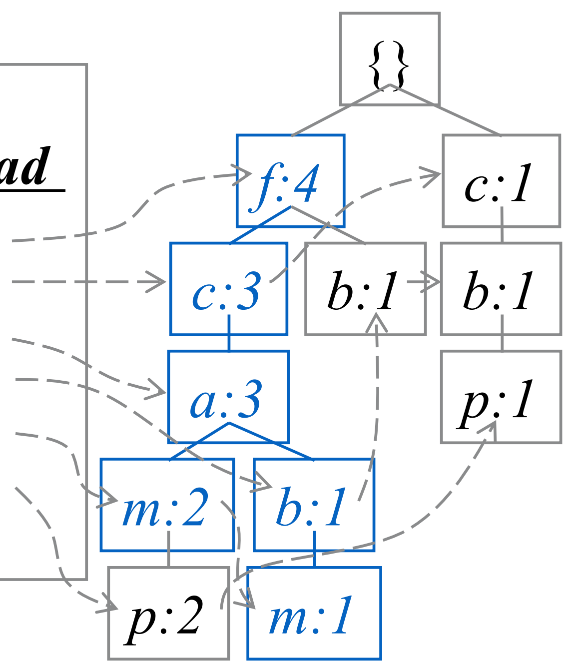
From Conditional Pattern-base to Conditional FP-Tree

For each pattern-base

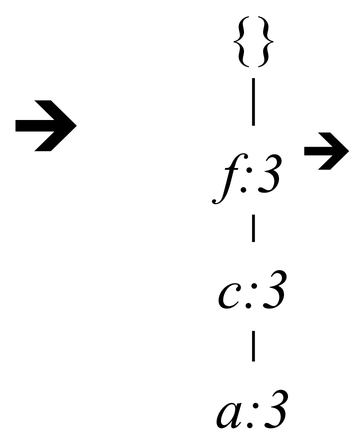
- Accumulate the count for each item in the base
- Construct the FP-tree for the frequent items of the pattern base

Header Table

Item	***frequency***	***head***
f	*4*	
c	*4*	
a	*3*	
b	*3*	
m	*3*	
p	*3*	



m-conditional **pattern base:**
fca:2, fcab:1



m-conditional **FP-tree**

All frequent patterns relate to *m*

m,

fm, cm, am,

fcm, fam, cam,

fcam

FPGrowth: Example

Part 1: Construct the FP-tree

FPGrowth Example

TID	Items
T100	I1, I2, I5
T101	I2, I4
T102	I2, I3
T103	I1, I2, I4
T104	I1, I3
T105	I2, I3
T106	I1, I3
T107	I1, I2, I3, I5
T108	I1, I2, I3

- Consider database D
- Let *minsup* = 2
- First scan is same as Apriori to derives 1-itemsets and their support counts
- Set of frequent items is sorted in order of descending support count
- L = {I2:7, I1:6, I3:6, I4:2, I5:2}

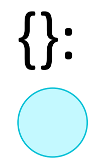
Construct FP-tree

- Create root of tree, labeled “null”
- Scan D a 2nd time (first scan was to create 1-itemsets and L)
- For each transaction, items are processed in L order (sorted order)
- Branch created for each transaction with items having their support count separated by colon
- Whenever same node is encountered in another transaction just increment support count of common node or prefix
- To facilitate tree traversal, an item header table is built so that each item points to its occurrences in the tree via a chain of node-links
- The problem of mining frequent patterns in D is transformed to mining the FP-tree

Construct FP-Tree: Start

TID	Items
T100	I1, I2, I5
T101	I2, I4
T102	I2, I3
T103	I1, I2, I4
T104	I1, I3
T105	I2, I3
T106	I1, I3
T107	I1, I2, I3, I5
T108	I1, I2, I3

Start, root = null

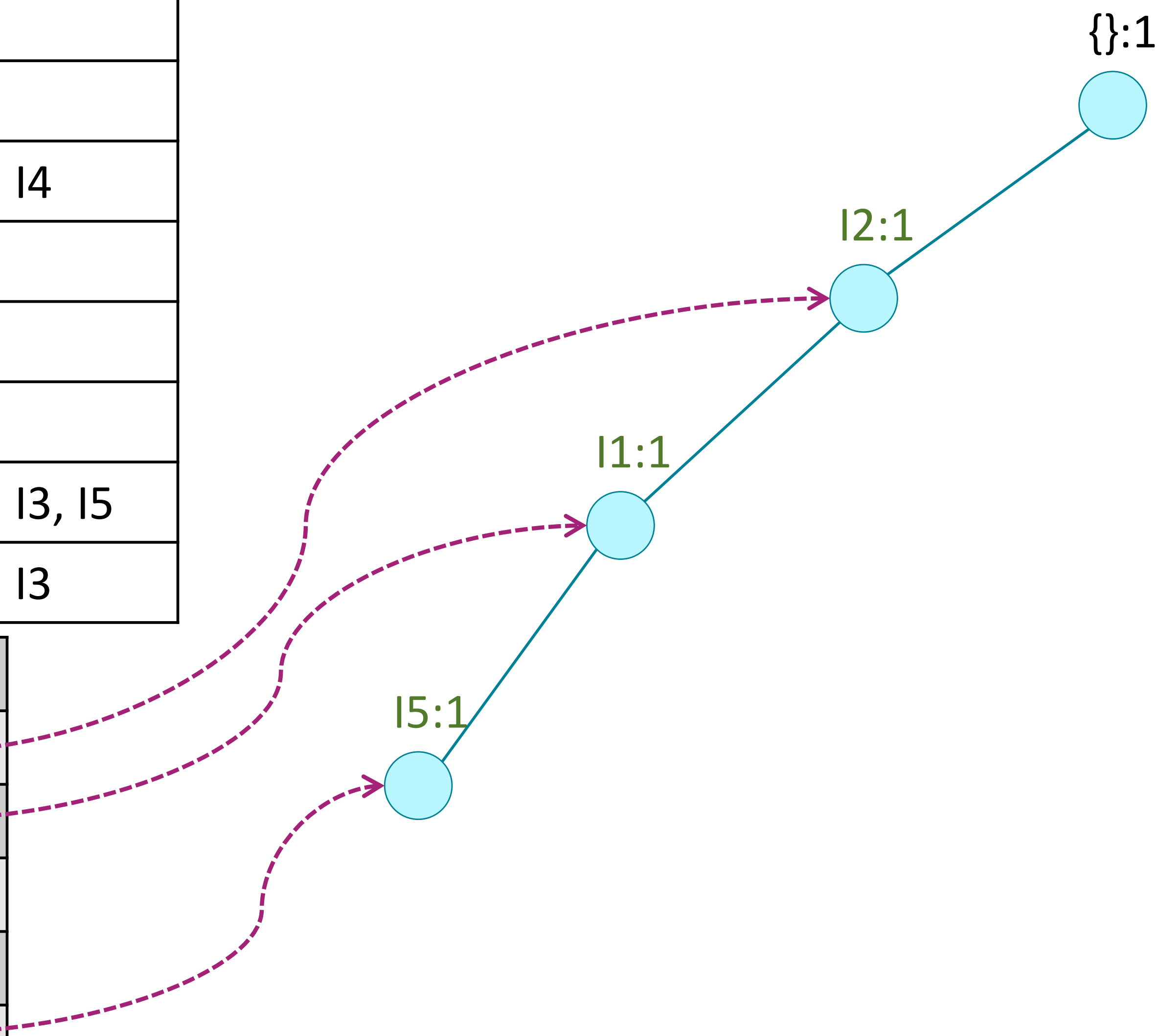


Header Table	
I2	
I1	
I3	
I4	
I5	

Construct FP-Tree: Trans. #1

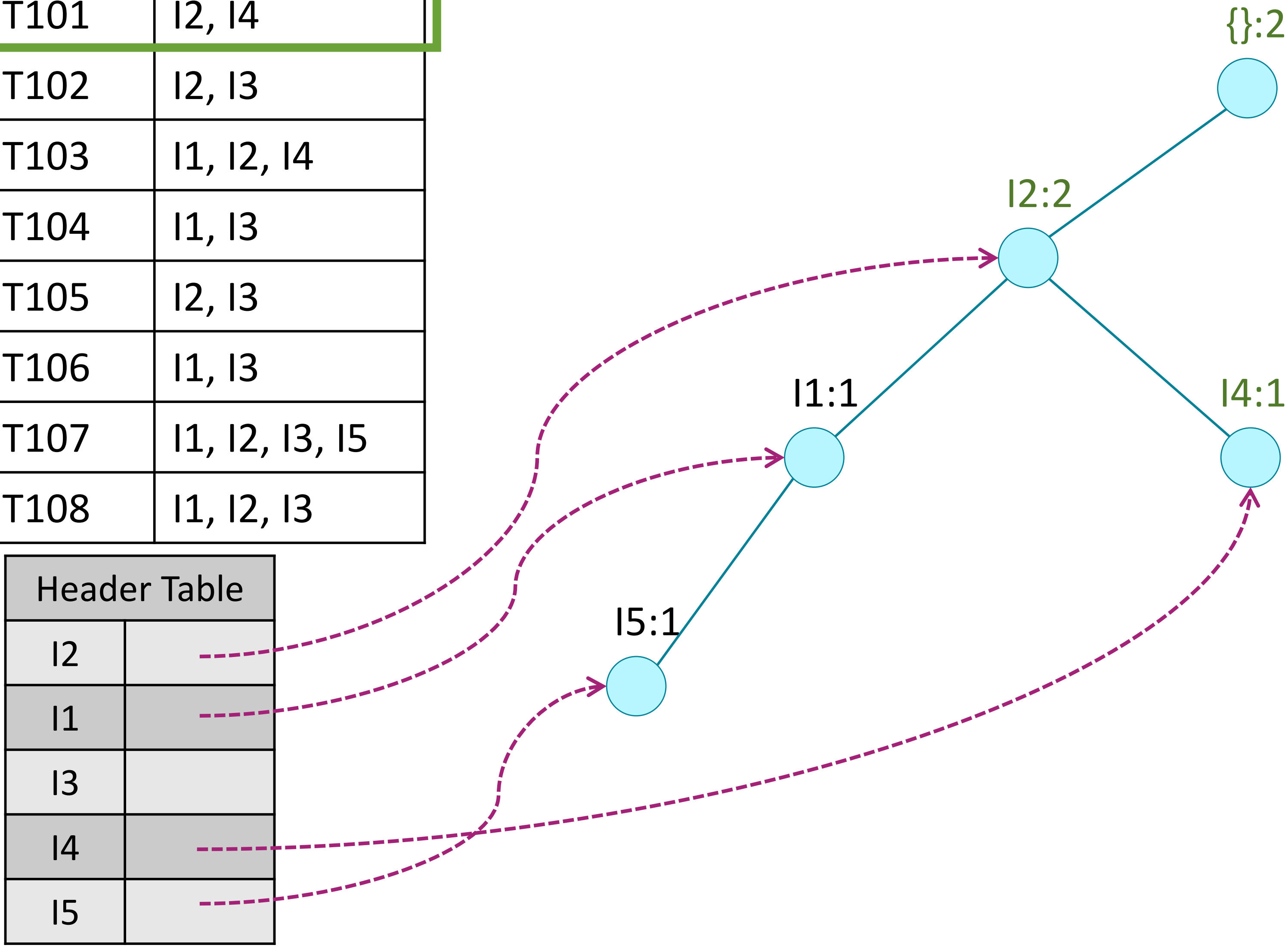
TID	Items
T100	I1, I2, I5
T101	I2, I4
T102	I2, I3
T103	I1, I2, I4
T104	I1, I3
T105	I2, I3
T106	I1, I3
T107	I1, I2, I3, I5
T108	I1, I2, I3

Header Table	
I2	
I1	
I3	
I4	
I5	



Construct FP-Tree: Trans. #2

TID	Items
T100	I1, I2, I5
T101	I2, I4
T102	I2, I3
T103	I1, I2, I4
T104	I1, I3
T105	I2, I3
T106	I1, I3
T107	I1, I2, I3, I5
T108	I1, I2, I3

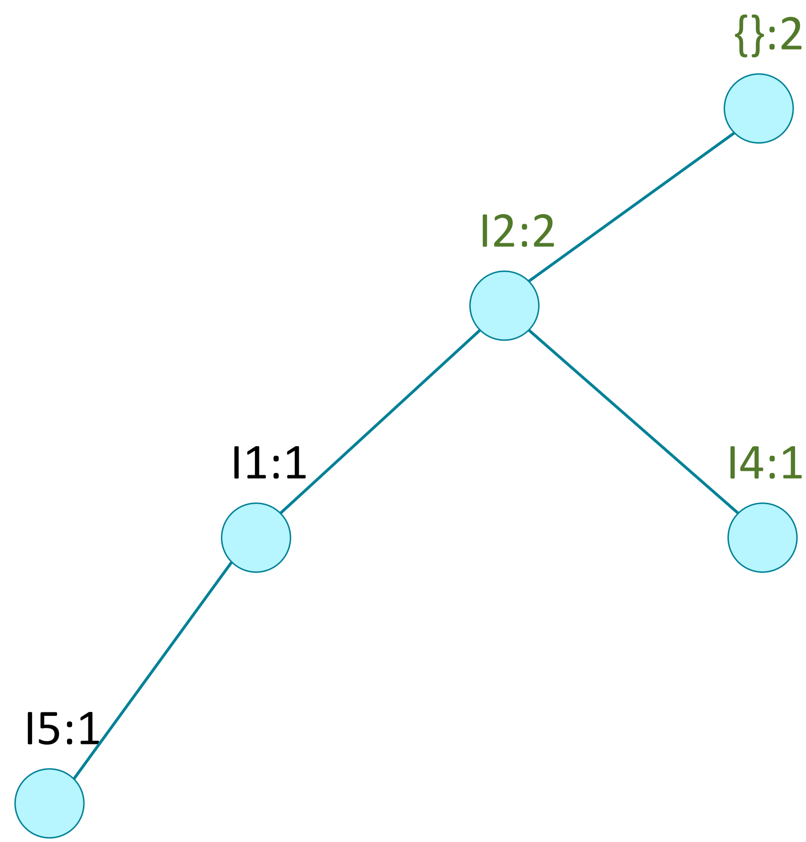


Construct FP-Tree: Trans. #2

TID	**Items**
T100	I1, I2, I5
T101	I2, I4
T102	I2, I3
T103	I1, I2, I4
T104	I1, I3
T105	I2, I3
T106	I1, I3
T107	I1, I2, I3, I5
T108	I1, I2, I3

Header Table	
I2	
I1	
I3	
I4	
I5	

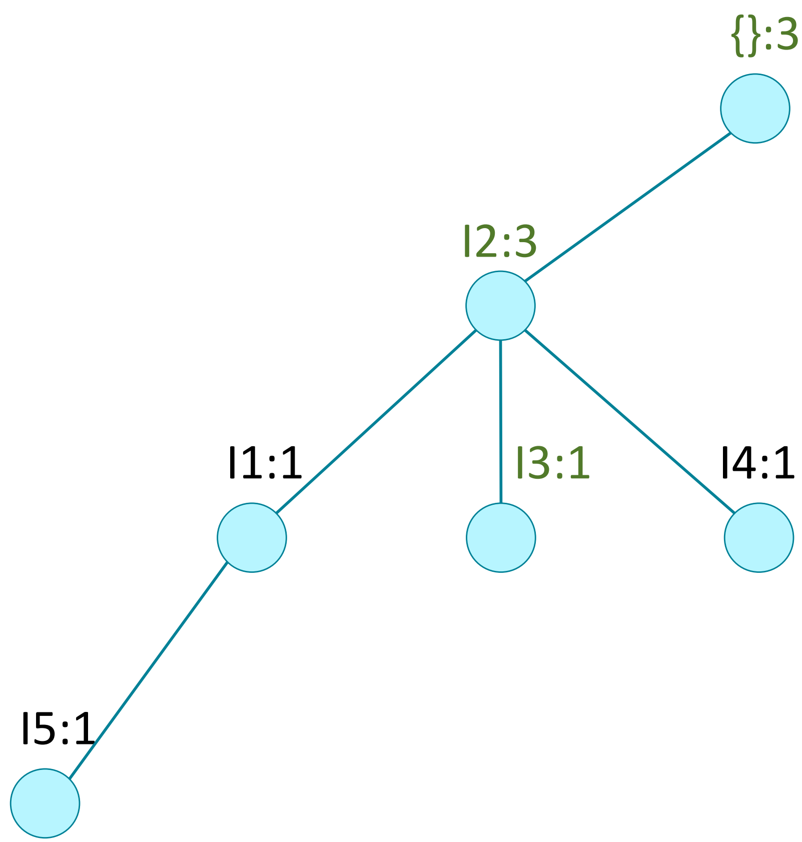
Remove header links to better see building of tree



Construct FP-Tree: Trans. #3

TID	Items
T100	I1, I2, I5
T101	I2, I4
T102	I2, I3
T103	I1, I2, I4
T104	I1, I3
T105	I2, I3
T106	I1, I3
T107	I1, I2, I3, I5
T108	I1, I2, I3

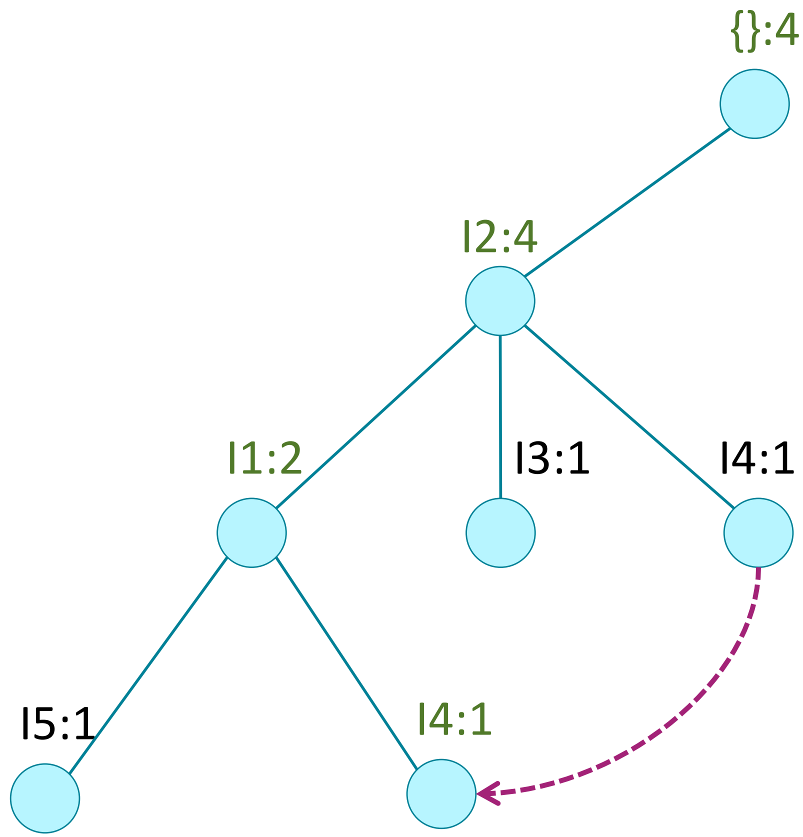
Header Table	
I2	
I1	
I3	
I4	
I5	



Construct FP-Tree: Trans. #4

TID	Items
T100	I1, I2, I5
T101	I2, I4
T102	I2, I3
T103	I1, I2, I4
T104	I1, I3
T105	I2, I3
T106	I1, I3
T107	I1, I2, I3, I5
T108	I1, I2, I3

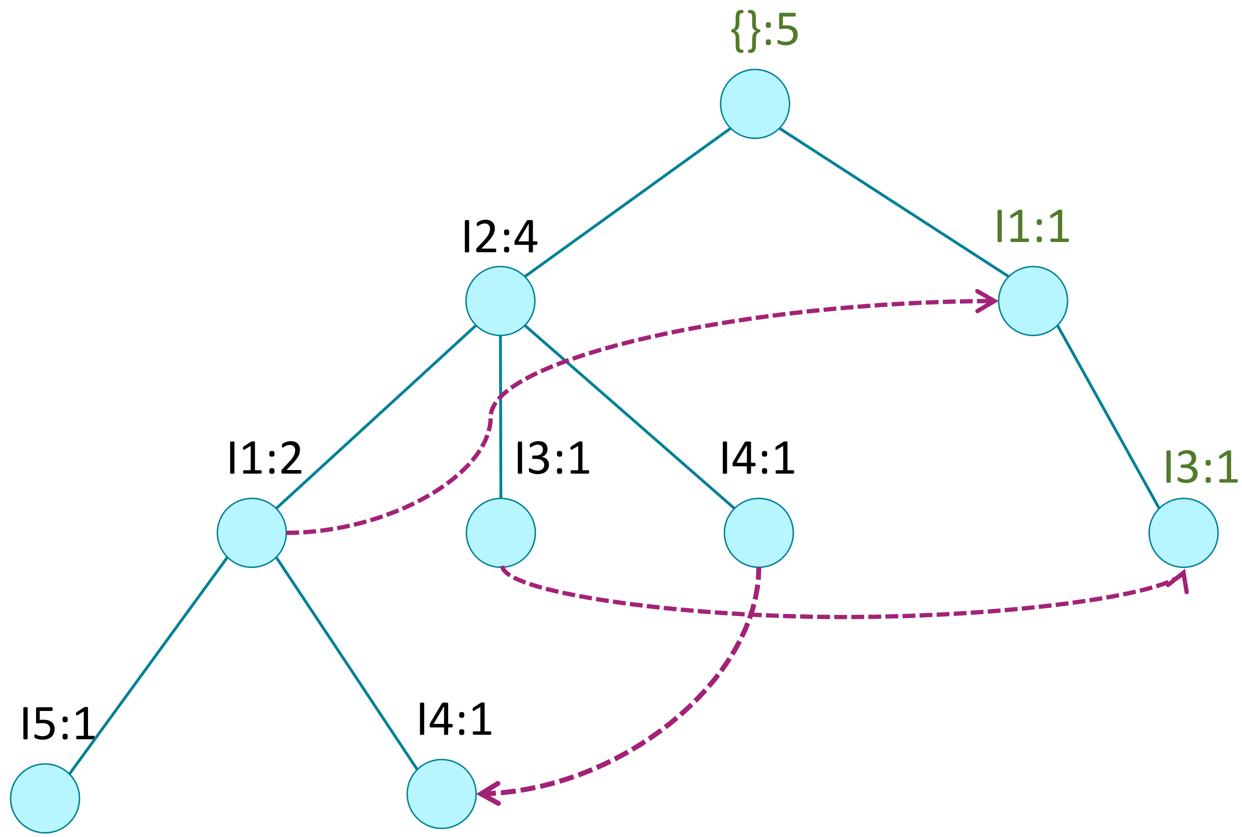
Header Table	
I2	
I1	
I3	
I4	
I5	



Construct FP-Tree: Trans. #5

TID	Items
T100	I1, I2, I5
T101	I2, I4
T102	I2, I3
T103	I1, I2, I4
T104	I1, I3
T105	I2, I3
T106	I1, I3
T107	I1, I2, I3, I5
T108	I1, I2, I3

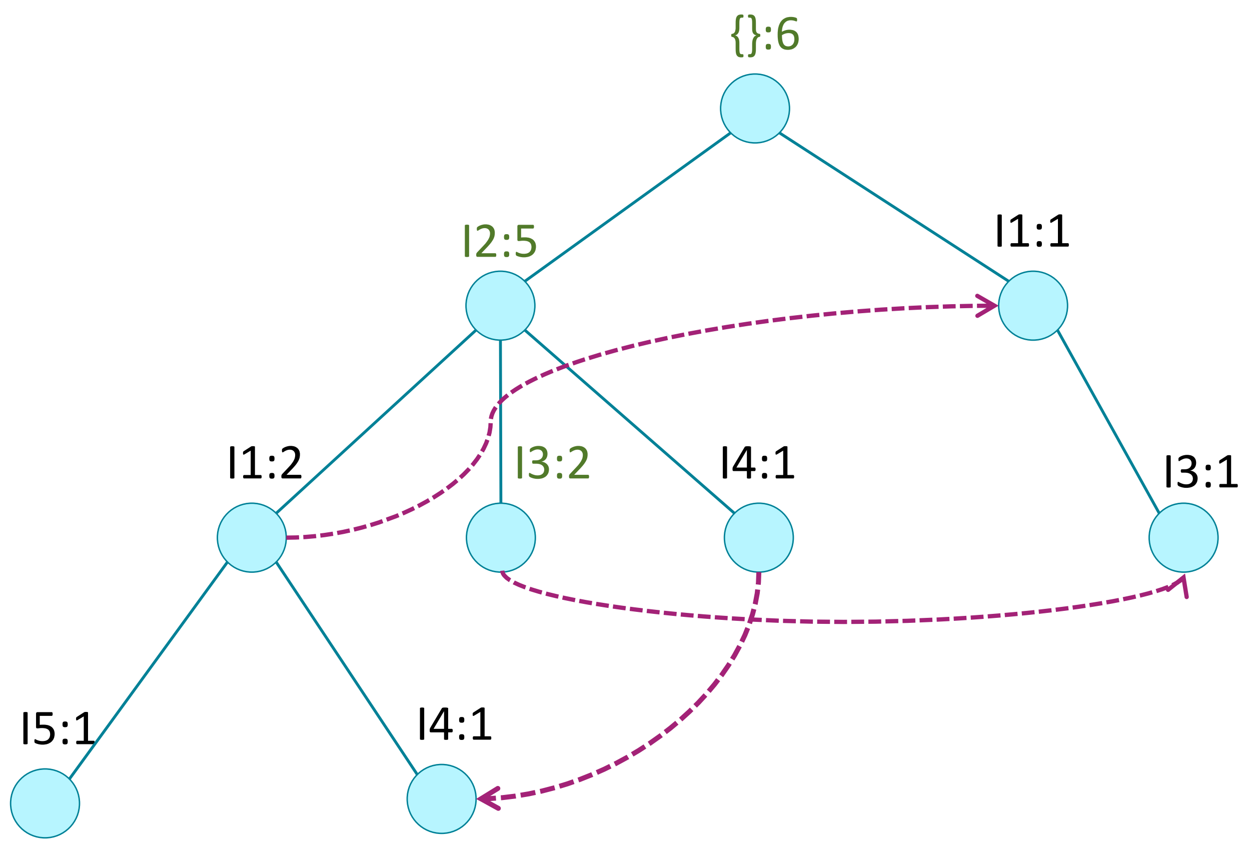
Header Table	
I2	
I1	
I3	
I4	
I5	



Construct FP-Tree: Trans. #6

TID	Items
T100	I1, I2, I5
T101	I2, I4
T102	I2, I3
T103	I1, I2, I4
T104	I1, I3
T105	I2, I3
T106	I1, I3
T107	I1, I2, I3, I5
T108	I1, I2, I3

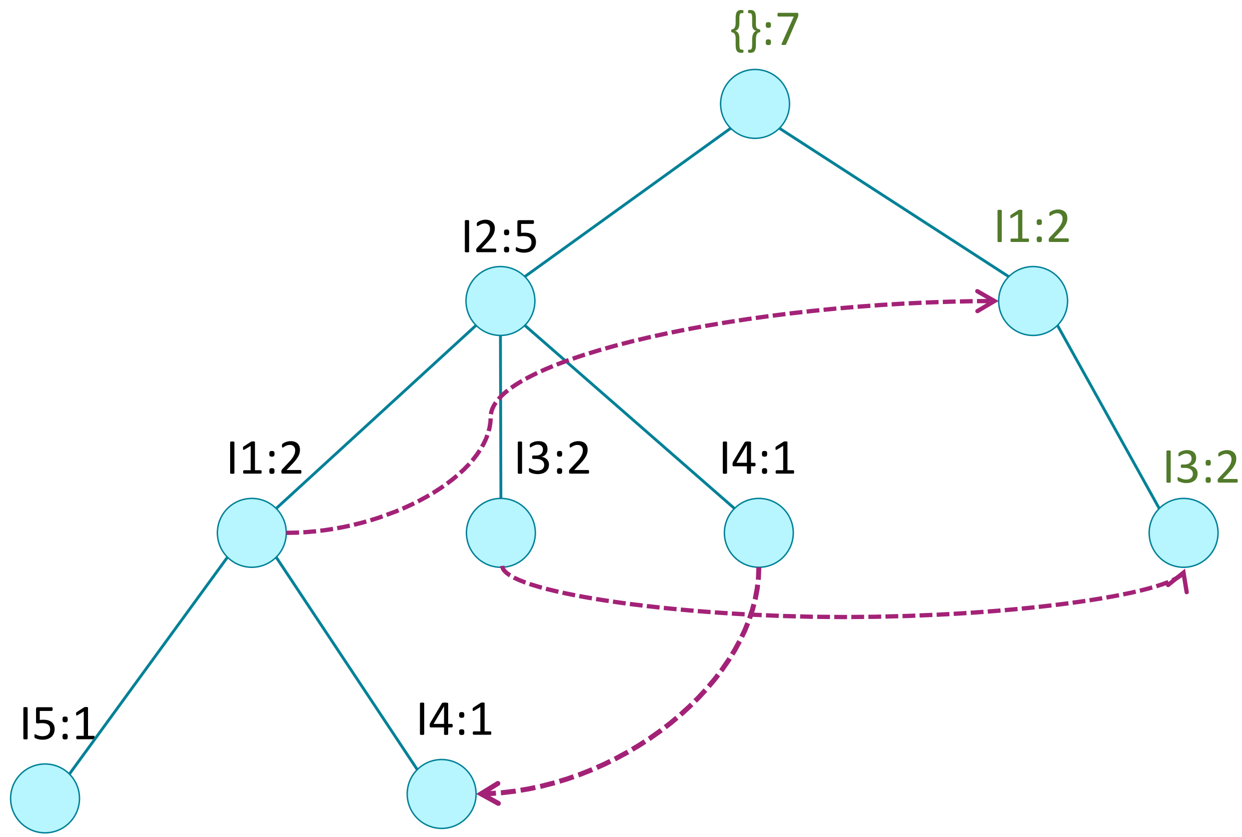
Header Table	
I2	
I1	
I3	
I4	
I5	



Construct FP-Tree: Trans. #7

TID	Items
T100	I1, I2, I5
T101	I2, I4
T102	I2, I3
T103	I1, I2, I4
T104	I1, I3
T105	I2, I3
T106	I1, I3
T107	I1, I2, I3, I5
T108	I1, I2, I3

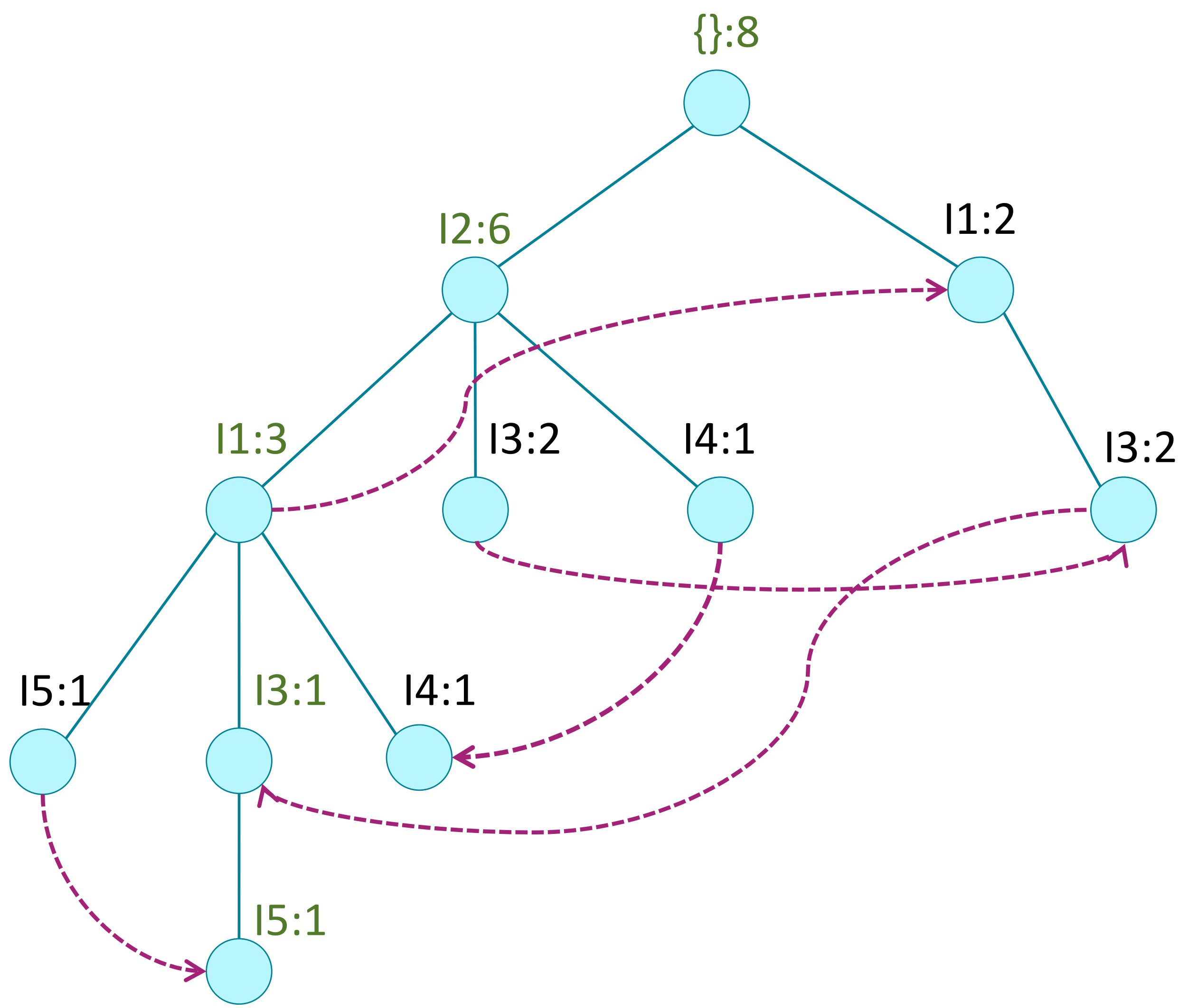
Header Table	
I2	
I1	
I3	
I4	
I5	



Construct FP-Tree: Trans. #8

TID	Items
T100	I1, I2, I5
T101	I2, I4
T102	I2, I3
T103	I1, I2, I4
T104	I1, I3
T105	I2, I3
T106	I1, I3
T107	I1, I2, I3, I5
T108	I1, I2, I3

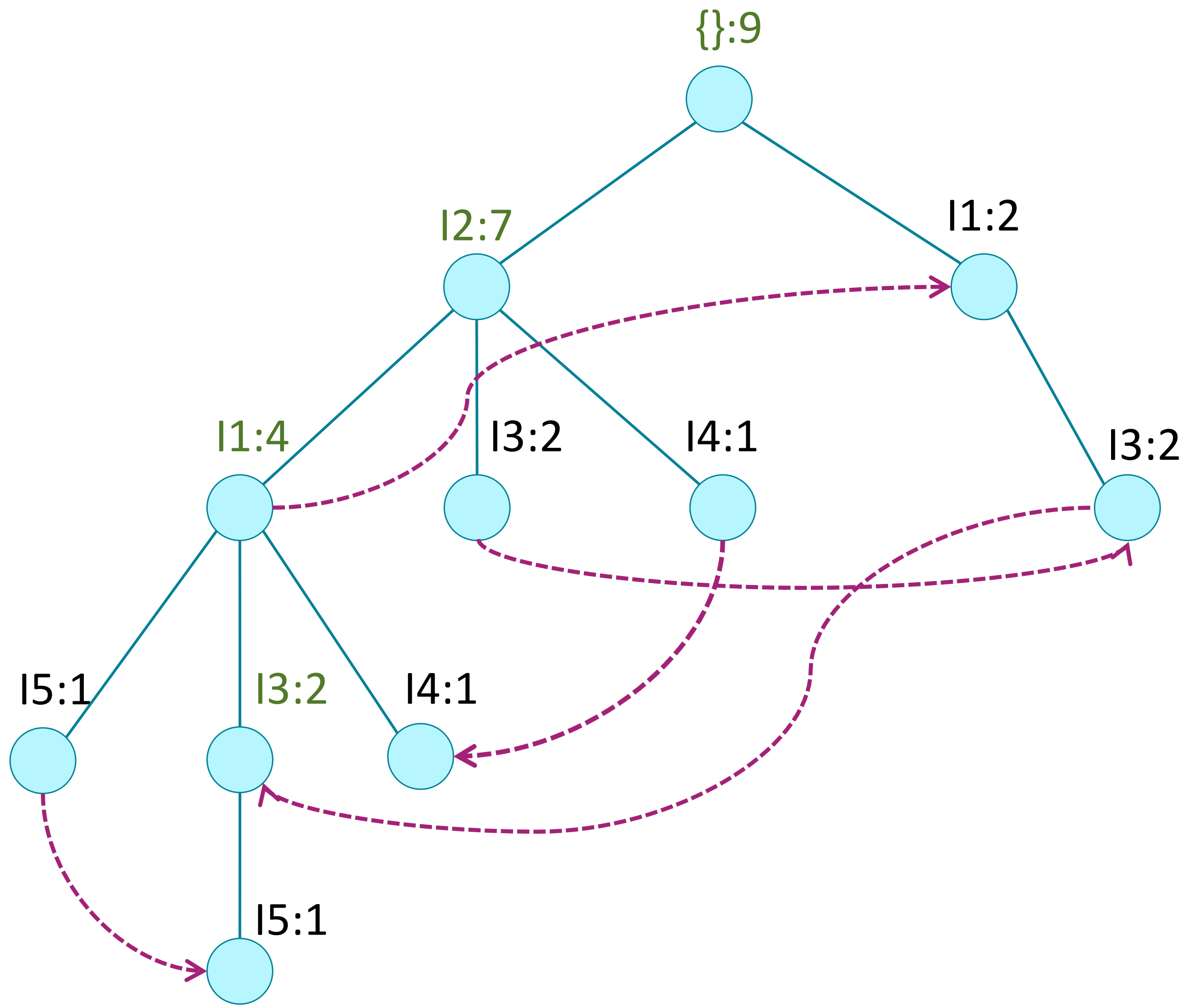
Header Table	
I2	
I1	
I3	
I4	
I5	



Construct FP-Tree: Trans. #9

TID	Items
T100	I1, I2, I5
T101	I2, I4
T102	I2, I3
T103	I1, I2, I4
T104	I1, I3
T105	I2, I3
T106	I1, I3
T107	I1, I2, I3, I5
T108	I1, I2, I3

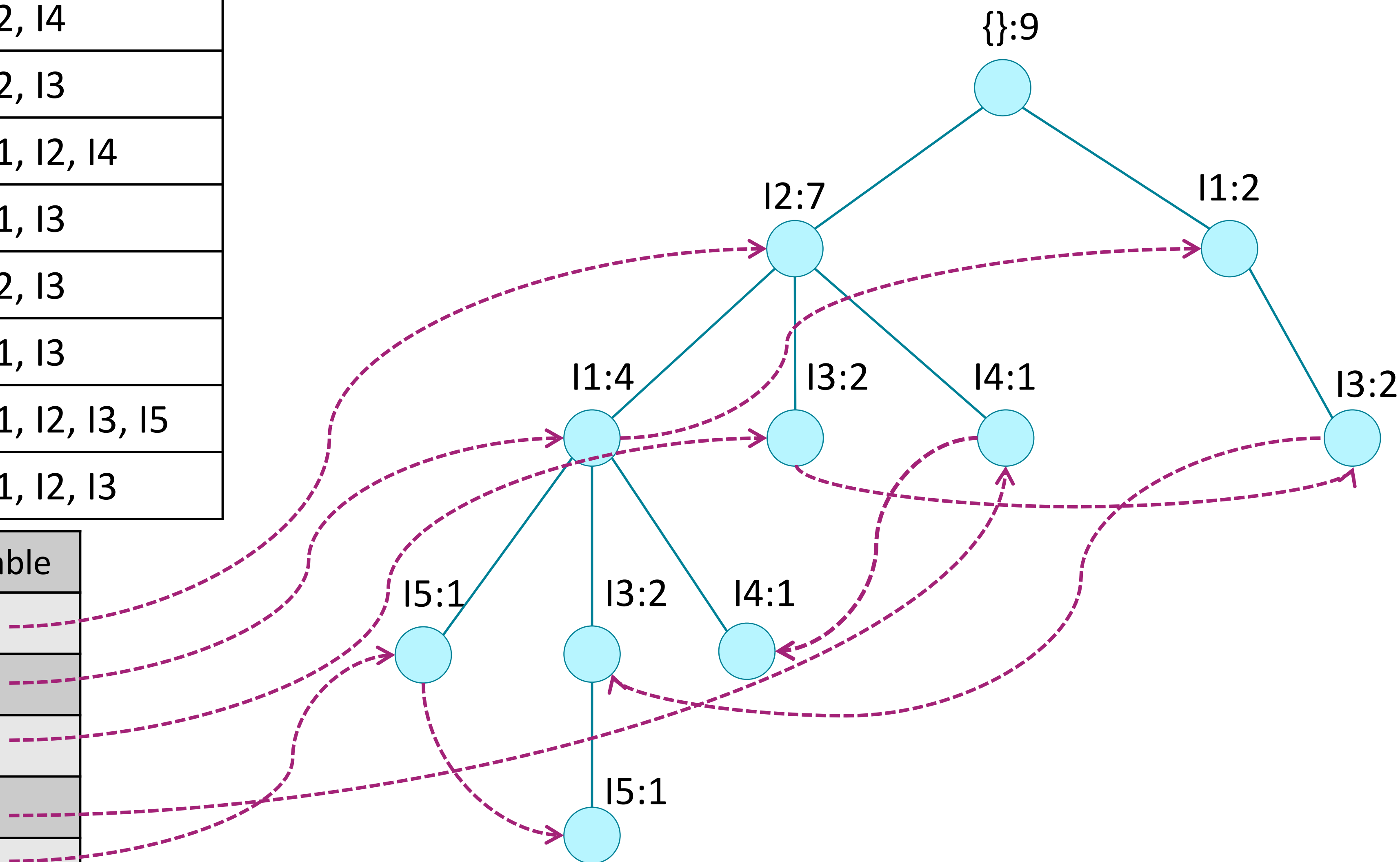
Header Table	
I2	
I1	
I3	
I4	
I5	



Construct FP-Tree: Complete

TID	Items
T100	I1, I2, I5
T101	I2, I4
T102	I2, I3
T103	I1, I2, I4
T104	I1, I3
T105	I2, I3
T106	I1, I3
T107	I1, I2, I3, I5
T108	I1, I2, I3

Header Table	
I2	
I1	
I3	
I4	
I5	



FPGrowth: Example

Part 2: Mine the FP-tree

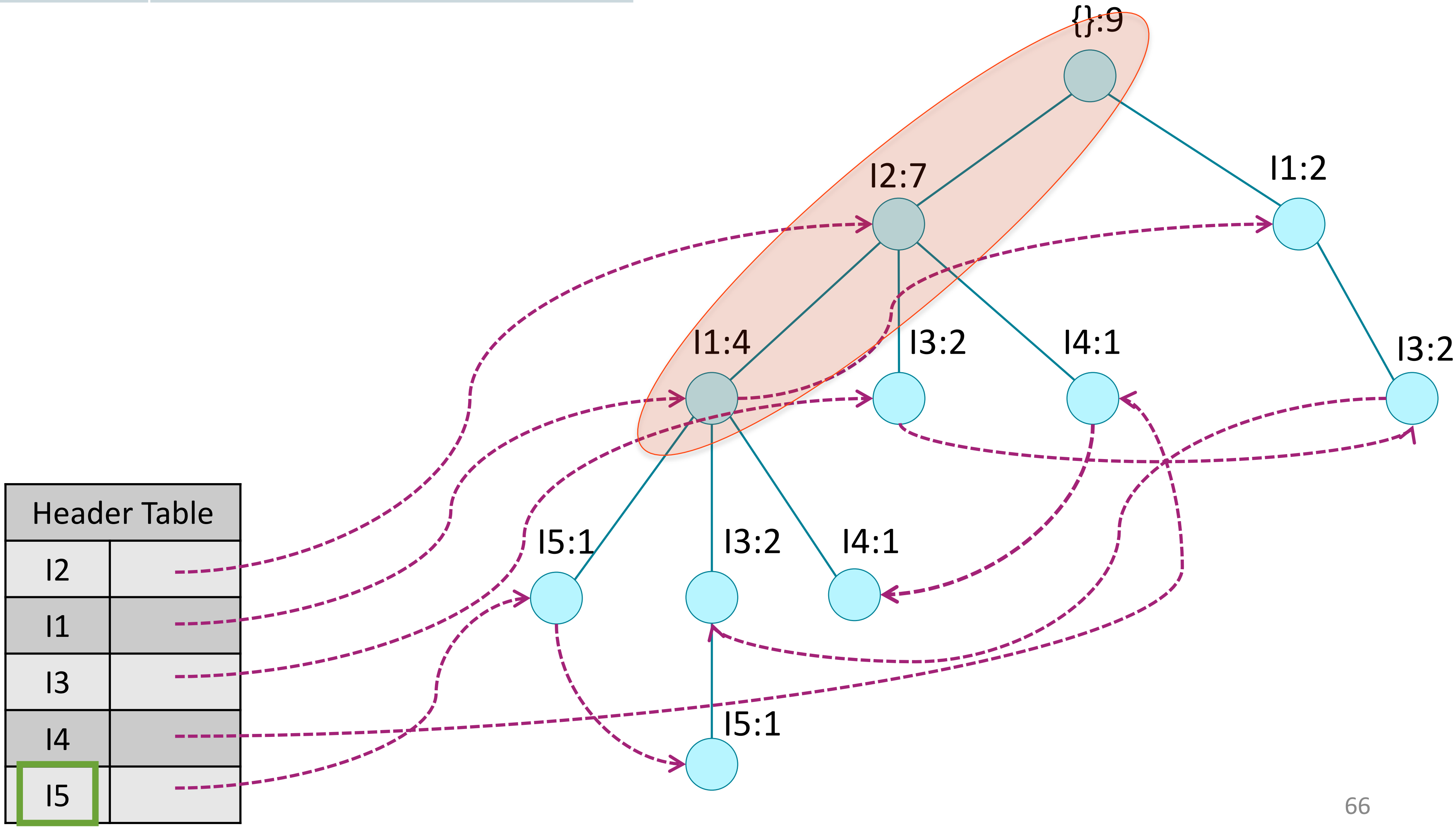
Mine FP-tree: Create Conditional Pattern Base

Steps

1. Start from each frequent length 1-pattern *i* (as an initial suffix pattern) in increasing order of support
2. Construct its conditional pattern base which consists of the set of prefix paths in the FP-tree co-occurring with suffix pattern (path from *i* in FP-tree to root)
3. Then, construct its conditional FP-tree & perform mining on such a tree
4. The pattern growth is achieved by concatenation of the suffix pattern with the frequent patterns generated from a conditional FP-tree
5. The union of all frequent patterns (generated by step 4) gives the required frequent itemset

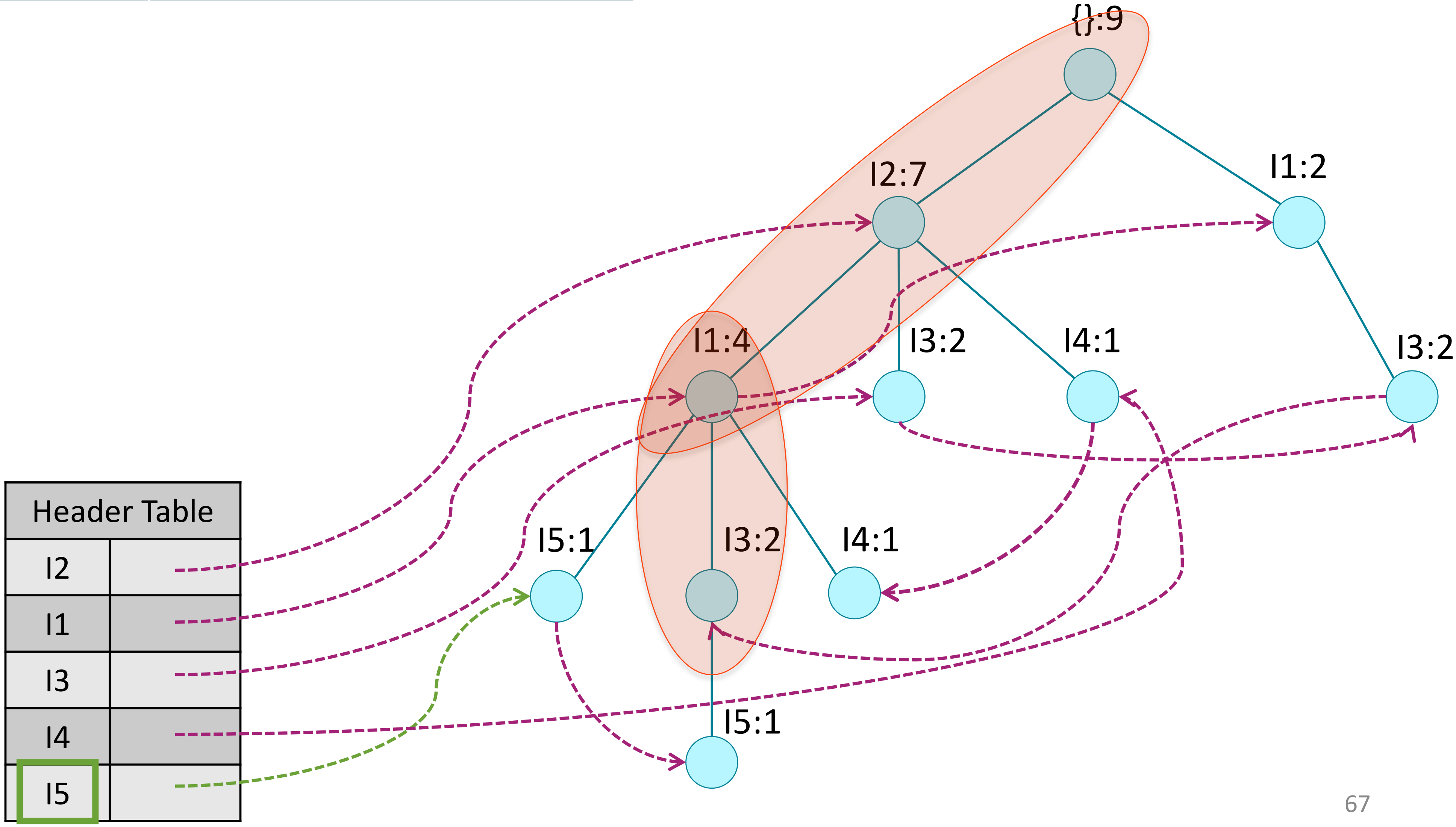
Construct Conditional Pattern Base

Item	Cond. Pattern
I5	{{I2 I1: 1}



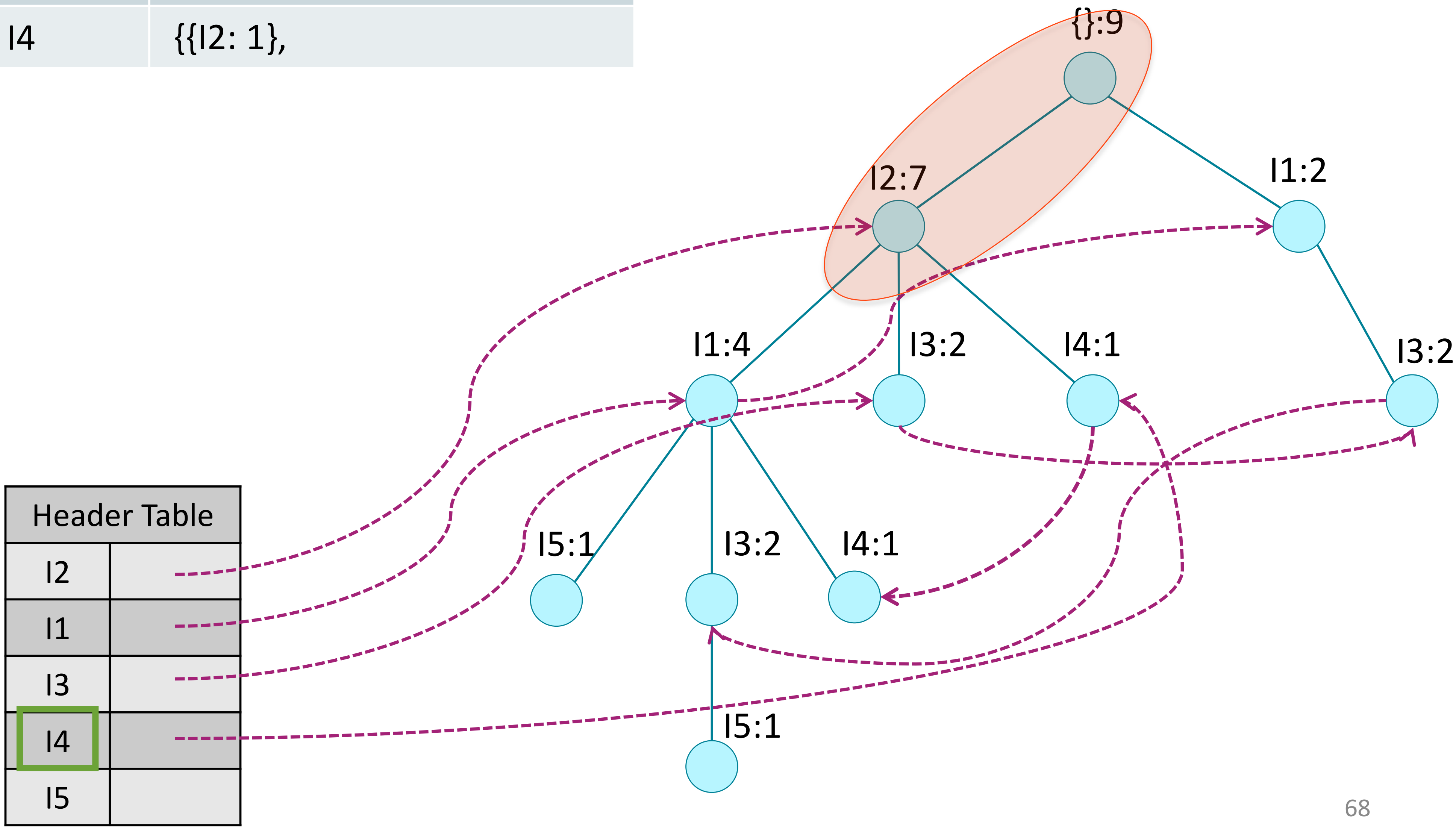
Construct Conditional Pattern Base

Item	Cond. Pattern
I5	{{I2 I1: 1}, {I2 I1 I3: 1}}



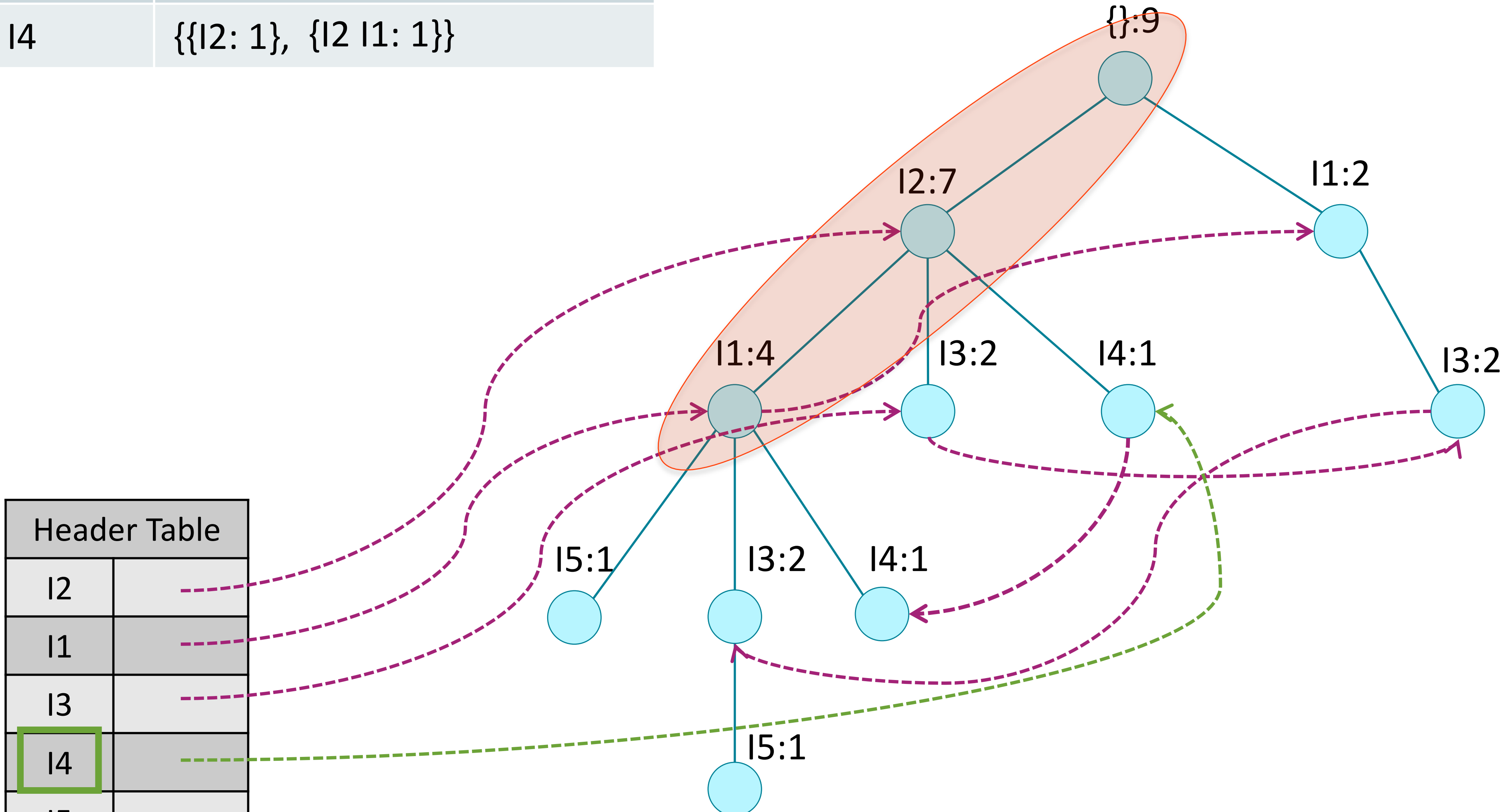
Construct Conditional Pattern Base

Item	Cond. Pattern
I5	{{I2 I1: 1}, {I2 I1 I3: 1}}
I4	{{I2: 1},



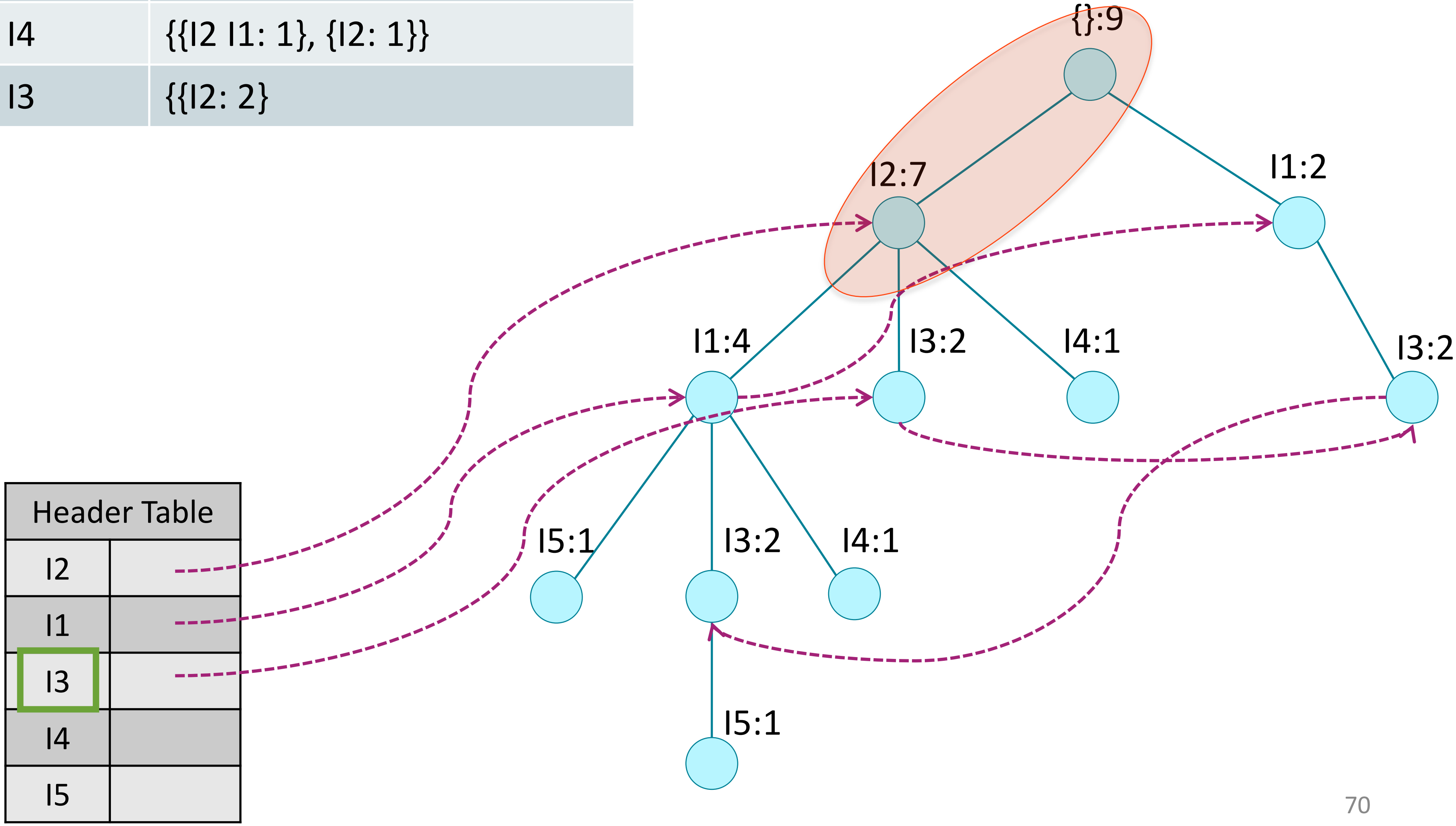
Construct Conditional Pattern Base

Item	Cond. Pattern
I5	{{I2 I1: 1}, {I2 I1 I3: 1}}
I4	{{I2: 1}, {I2 I1: 1}}



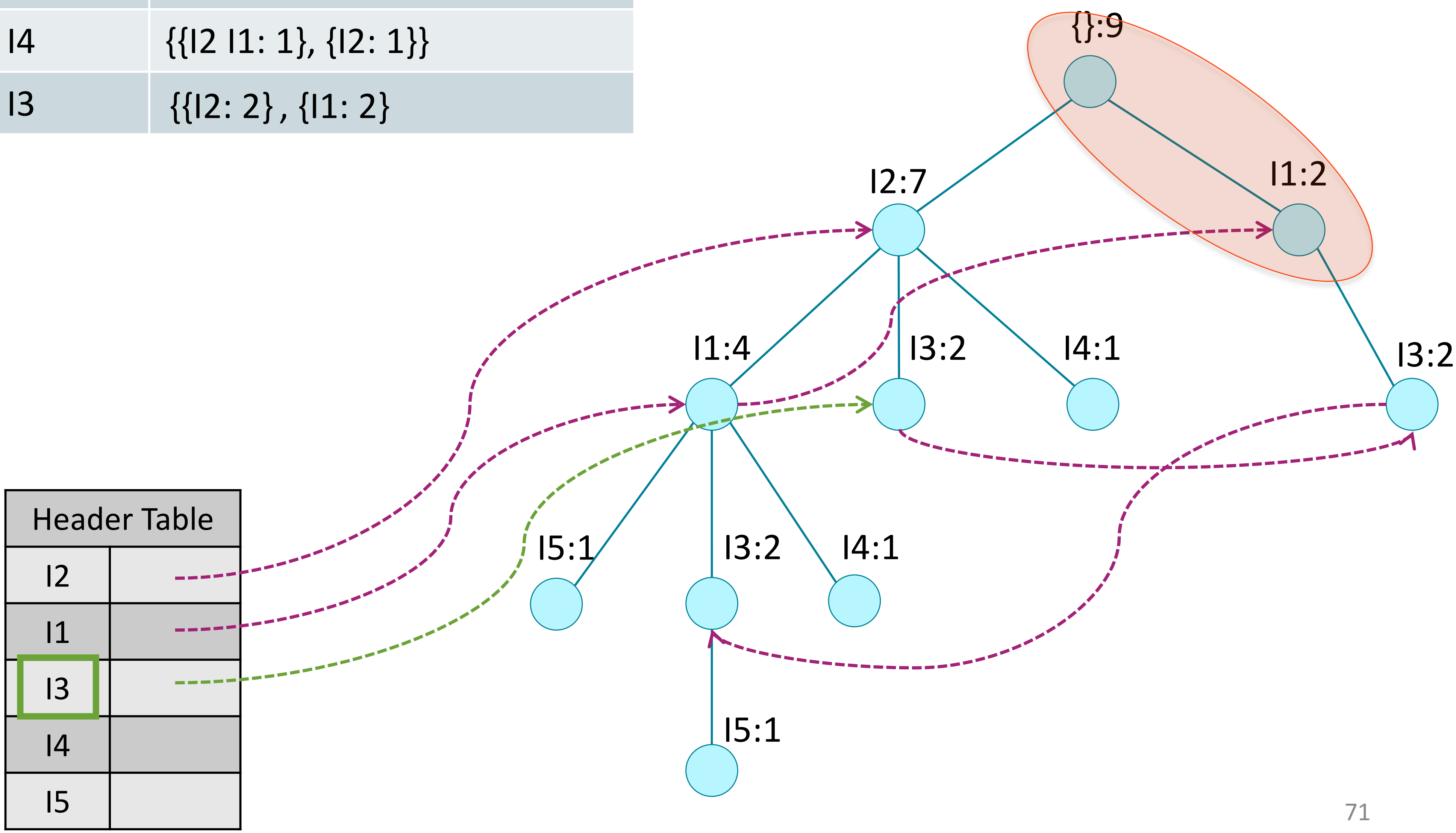
Construct Conditional Pattern Base

Item	Cond. Pattern
I5	{{I2 I1: 1}, {I2 I1 I3: 1}}
I4	{{I2 I1: 1}, {I2: 1}}
I3	{{I2: 2}



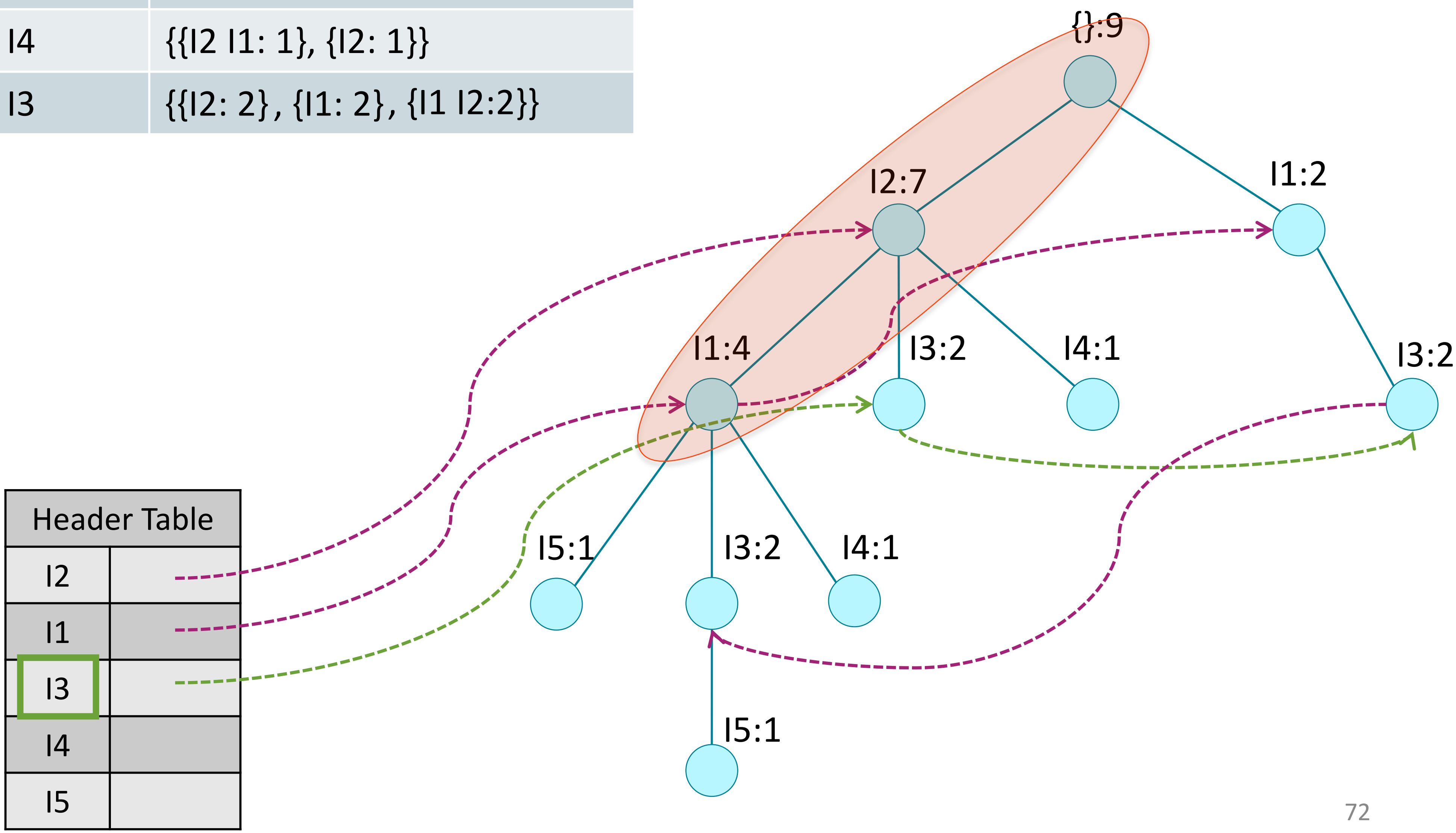
Construct Conditional Pattern Base

Item	Cond. Pattern
I5	{{I2 I1: 1}, {I2 I1 I3: 1}}
I4	{{I2 I1: 1}, {I2: 1}}
I3	{{I2: 2}, {I1: 2}



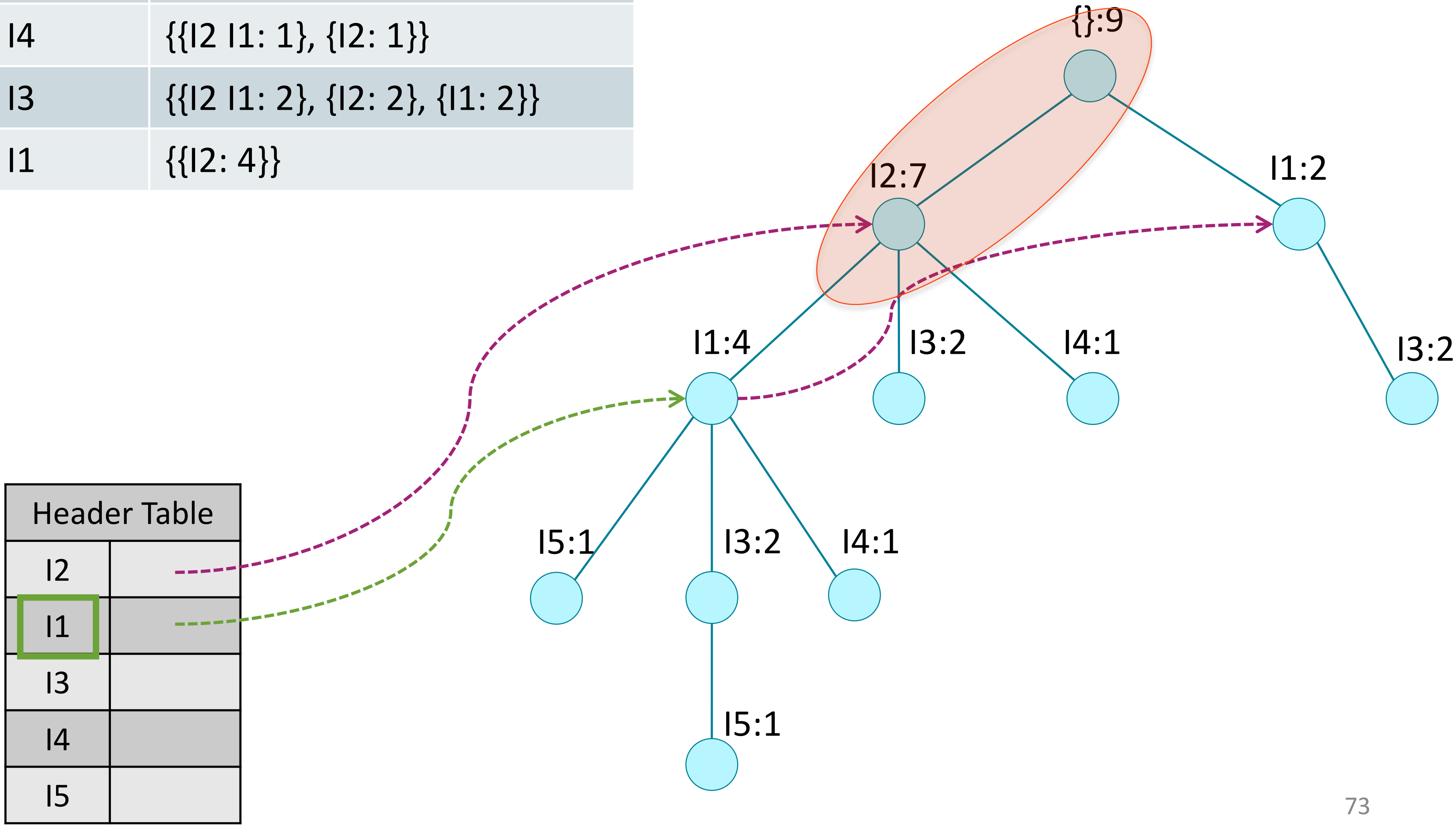
Construct Conditional Pattern Base

Item	Cond. Pattern
I5	{{I2 I1: 1}, {I2 I1 I3: 1}}
I4	{{I2 I1: 1}, {I2: 1}}
I3	{{I2: 2}, {I1: 2}, {I1 I2:2}}



Construct Conditional Pattern Base

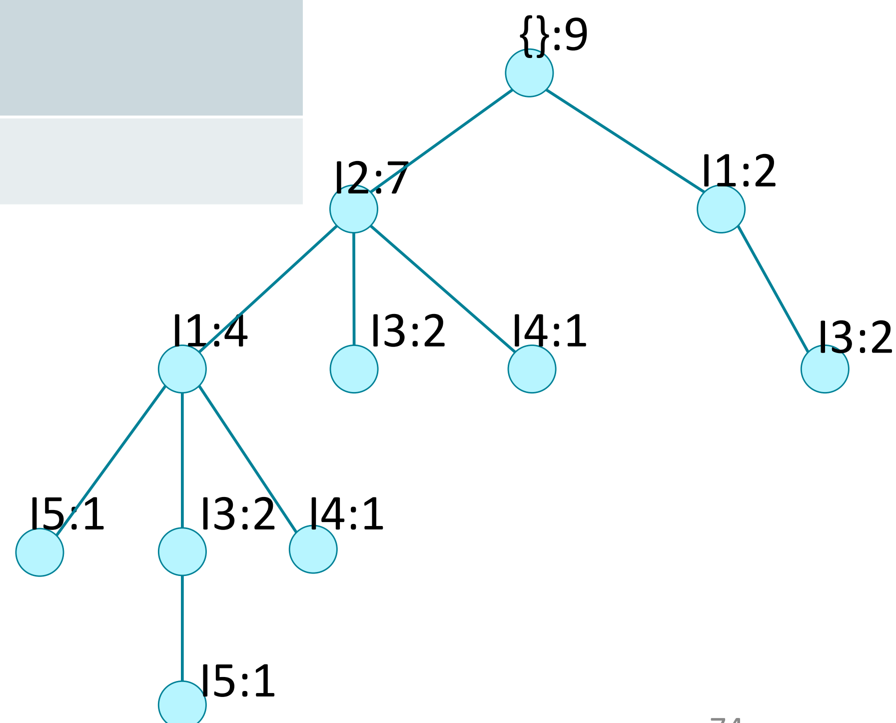
Item	Cond. Pattern
I5	{{I2 I1: 1}, {I2 I1 I3: 1}}
I4	{{I2 I1: 1}, {I2: 1}}
I3	{{I2 I1: 2}, {I2: 2}, {I1: 2}}
I1	{{I2: 4}}



Cond. Pattern Base -> Create Cond. FP-Tree

Item	Cond. Pattern	Cond. FP-tree	Frequent Pattern
I5	{(I2 I1: 1), (I2 I1 I3: 1)}		
I4	{(I2 I1: 1), (I2: 1)}		
I3	{(I2 I1: 2), (I2: 2), (I1: 2)}		
I1	{(I2: 4)}		

- Create Cond. FP-tree using conditional patterns
- Frequent pattern with each suffix is generated by considering all possible combinations of the item and FP-tree



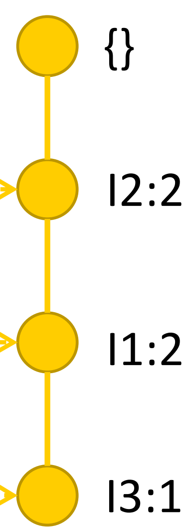
FPGrowth: Example

Part 3 – Create Conditional FP-tree and generate patterns

Cond. Pattern Base -> Create Cond. FP-Tree

Item	Cond. Pattern	Cond. FP-tree	Frequent Pattern
I5	{(I2 I1: 1), (I2 I1 I3: 1)}		
I4	{(I2 I1: 1), (I2: 1)}		
I3	{(I2 I1: 2), (I2: 2), (I1: 2)}		
I1	{(I2: 4)}		

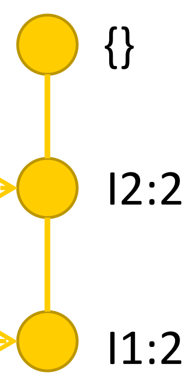
Item	Sup	link
I2	2	
I1	2	
I3	1	



Cond. Pattern Base -> Create Cond. FP-Tree

Item	Cond. Pattern	Cond. FP-tree	Frequent Pattern
I5	{(I2 I1: 1), (I2 I1 I3: 1)}	<I2: 2, I1: 2>	{I2 I5: 2}, {I1 15: 2}, {I2 I1 I5: 2}
I4	{(I2 I1: 1), (I2: 1)}		
I3	{(I2 I1: 2), (I2: 2), (I1: 2)}		
I1	{(I2: 4)}		

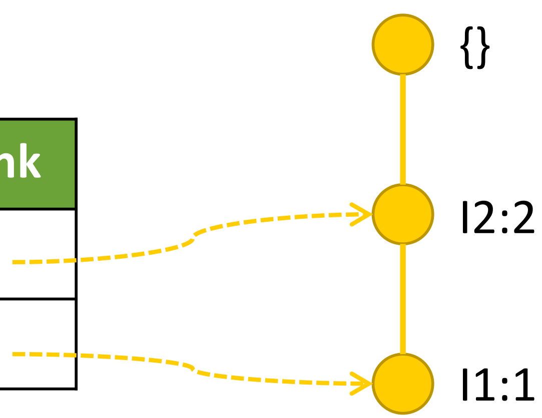
Item	Sup	link
I2	2	
I1	2	
I3	1	



Cond. Pattern Base -> Create Cond. FP-Tree

Item	Cond. Pattern	Cond. FP-tree	Frequent Pattern
I5	{(I2 I1: 1), (I2 I1 I3: 1)}	<I2:2, I1:2>	{I2 I5: 2}, {I1 I5: 2}, {I2 I1 I5: 2}
I4	{(I2 I1: 1), (I2: 1)}		
I3	{(I2 I1: 2), (I2: 2), (I1: 2)}		
I1	{(I2: 4)}		

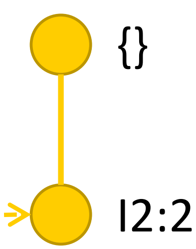
Item	Sup	link
I2	2	
I1	1	



Cond. Pattern Base -> Create Cond. FP-Tree

Item	Cond. Pattern	Cond. FP-tree	Frequent Pattern
I5	{(I2 I1: 1), (I2 I1 I3: 1)}	<I2:2, I1:2>	{I2 I5: 2}, {I1 I5: 2}, {I2 I1 I5: 2}
I4	{(I2 I1: 1), (I2: 1)}	<I2: 2>	{I2 I4: 2}
I3	{(I2 I1: 2), (I2: 2), (I1: 2)}		
I1	{(I2: 4)}		

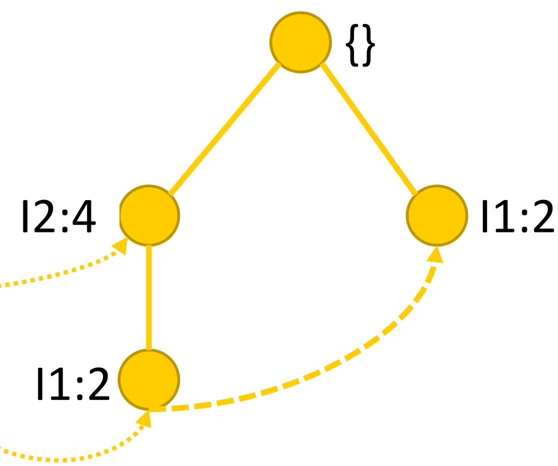
Item	Sup	link
I2	2	
I1	1	



Cond. Pattern Base -> Create Cond. FP-Tree

Item	Cond. Pattern	Cond. FP-tree	Frequent Pattern
I5	{(I2 I1: 1), (I2 I1 I3: 1)}	<I2:2, I1:2>	{I2 I5: 2}, {I1 I5: 2}, {I2 I1 I5: 2}
I4	{(I2 I1: 1), (I2: 1)}	<I2: 2>	{I2 I4: 2}
I3	{(I2 I1: 2), (I2: 2), (I1: 2)}	<I2: 4, I1: 2>, <I1: 2>	
I1	{(I2: 4)}		

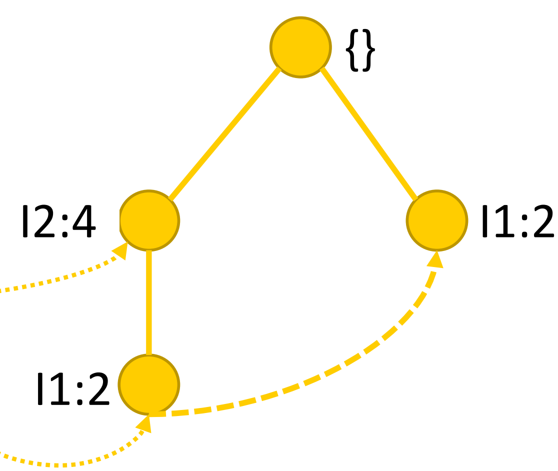
Item	Sup	link
I2	4	
I1	4	



Cond. Pattern Base -> Create Cond. FP-Tree

Item	Cond. Pattern	Cond. FP-tree	Frequent Pattern
I5	{(I2 I1: 1), (I2 I1 I3: 1)}	<I2:2, I1:2>	{I2 I5: 2}, {I1 I5: 2}, {I2 I1 I5: 2}
I4	{(I2 I1: 1), (I2: 1)}	<I2: 2>	{I2 I4: 2}
I3	{(I2 I1: 2), (I2: 2), (I1: 2)}	<I2: 4, I1: 2>, <I1: 2>	{{I2 I1 I3: 2}, {I1 I3: 4}, {I2 I3: 4}}
I1	{(I2: 4)}		

Item	Sup	link
I2	4	
I1	4	



Cond. Pattern Base -> Create Cond. FP-Tree

Item	Cond. Pattern	Cond. FP-tree	Frequent Pattern
I5	{(I2 I1: 1), (I2 I1 I3: 1)}	<I2:2, I1:2>	{I2 I5: 2}, {I1 I5: 2}, {I2 I1 I5: 2}
I4	{(I2 I1: 1), (I2: 1)}	<I2: 2>	{I2 I4: 2}
I3	{(I2 I1: 2), (I2: 2), (I1: 2)}	<I2: 4, I1:2>, <I1: 2>	{I2 I3: 4}, {I1 I3: 4}, {I2 I1 I3: 2}
I1	{(I2: 4)}	<I2: 4>	{I2, I1: 4}

Frequent Itemsets Identified:

{ 1, 2, 3, 4, 5, 12, 13, 15, 23, 24, 25, 123, 125 }

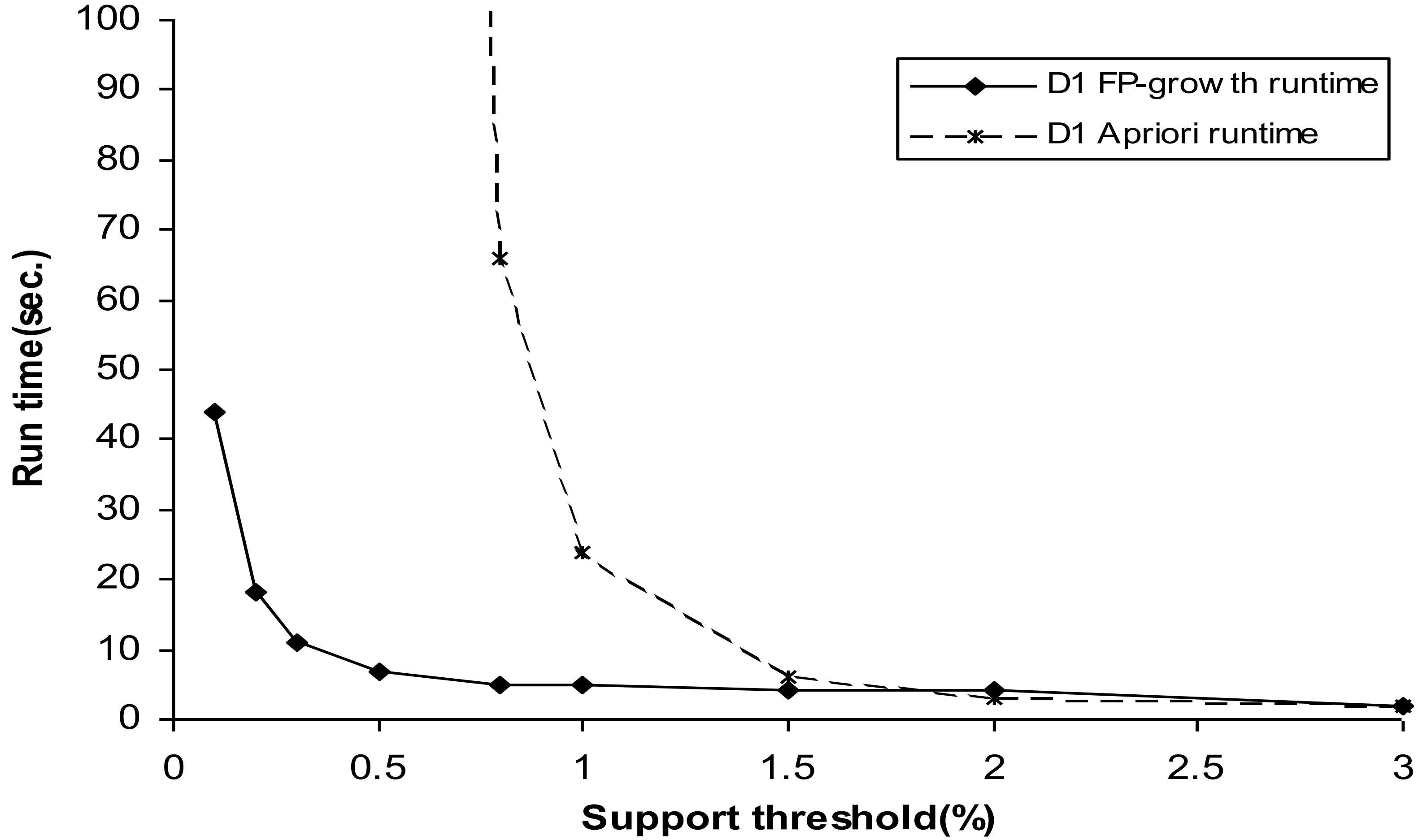
Benefits of FP-tree

- Completeness
  - Preserve complete information for frequent pattern mining
  - Never break a long pattern of any transaction
- Compactness
  - Reduce irrelevant info – infrequent items are gone
  - Items in frequency descending order: the more frequently occurring, the more likely to be shared
  - Never larger than original database

Benefits of FPGrowth

- Performance study shows
  - FPGrowth is an order of magnitude faster than Apriori, also faster than tree-projection
- Reasoning
  - no candidate generation, no candidate test
  - use compact data structure
  - eliminate repeated database scan
  - basic operation is counting and FP-tree building

FPGrowth vs. Apriori



Other Improvements in Mining

- AFOPT (Liu et al., KDD 2003)
  - A “push-right” method for mining condensed frequent pattern (CFP) trees
- Carpenter (Pan et al., KDD 2003)
  - Mine data sets with small rows but numerous columns
  - Construct a row-enumeration tree for efficient mining
- Fpgrowth+ (Grahne and Zhu, FIMI 2003)
  - Efficiently using prefix trees, open-source implementation
  - ICDM 2003
- TD-Close (Liu et al., SDM 2006)

Additional Improvements in Mining

- Mining closed frequent itemsets and max-patterns
  - CLOSET (DMKD'00), FPclose, and FPMax (Grahne & Zhu, Fimi'03)
- Mining sequential patterns
  - PrefixSpan (ICDE'01), CloSpan (SDM'03), BIDE (ICDE'04)
- Mining graph patterns
  - gSpan (ICDM'02), CloseGraph (KDD'03)
- Constraint-based mining of frequent patterns
  - Convertible constraints (ICDE'01), gPrune (PAKDD'03)
- Computing iceberg data cubes with complex measures
  - H-tree, H-cubing, and Star-cubing (SIGMOD'01, VLDB'03)
- Pattern-growth-based Clustering
  - MaPle (Pei, et al., ICDM'03)
- Pattern-Growth-Based Classification
  - Mining frequent and discriminative patterns (Cheng, et al, ICDE'07)