Data Mining Tutorial - Frequent Pattern Mining

Kyuseok Shim
Seoul National University
http://ee.snu.ac.kr/~shim

Frequent Pattern Mining

- Association rules
  - Apriori
  - FP-tree
- Sequential patterns
  - Apriori
  - PrefixSpan

Association Rules

구매 번호	구매 상품들
1	{라면, 우유, 오렌지 쥬스, 커피}
2	{라면, 우유, 소시지}
3	{라면, 우유, 커피}
4	{오렌지 쥬스, 비누, 샴푸}

- 데이터 상호간의 연관 규칙을 찾아내는 기술
- '{라면, 우유}->{커피}'
  - 라면과 우유를 산 사람은 커피도 같이 산다
  - 지지도 (support)
    - 전체 트랜잭션의 중에서 그 규칙을 가지고 있는 트랜잭션의 퍼센트
    - 50% – 네 가지 트랜잭션 중 1번과 3번 소비자의 구매한 물건들에 들어 있는 규칙
  - 신뢰도 (confidence)
    - 규칙의 왼쪽에 있는 것들을 산 사람들 중에서 오른쪽에 있는 물건들을 모두 산 사람들의 퍼센트
    - 66.7% – 라면과 우유를 산 사람들은 세 사람인데 그 중에서 커피를 산 사람은 두 사람이므로

사용 사례

- 고객들의 물품 구매 패턴을 분석한 결과에 기반하여
  - 연관 물품 쿠폰이나 할인 행사 제공
  - 온라인 서점에서 다른 구매자들이 구매한 책 정보를 함께 제공
  - 더 높은 가격의 상품 추천 (up-selling)
  - 백화점의 Package 구매 상품 조합 결정, 물건 진열 순서 결정 등
  - 상품 카탈로그 디자인
  - 백화점에서는 물건 진열 (Shelf planning)

사용 사례

- 같이 구매하는 경우가 많을 때 그 중의 어느 물건을 사면 다른 물건을 추천
  - Package로 물건 몇 개를 함께 포장해서 팔 때에 product mix를 결정함
  - 소비자의 그룹에 따른 brand royalty를 알아내고 상품추천
  - Cross-selling (교차 판매) – 서로 다른 카타고리 상품을 추천하여 판매
    - 자동차 보험과 생명보험을 함께 판매하는 온라인 보험회사에서 10 억짜리 생명보험을 가입한 사람에게 자동차보험 대물배상 3억원 추천
    - 11번가에서 디지털카메라를 구매할 때에 명품지갑 추천
  - Up-selling – 1 억짜리 생명보험을 구매할 때에 다른 고객들의 빈번한 패턴을 이용 10억짜리를 구매할 고객으로 판단되어 10억 짜리 생명보험을 추천함

사용 사례

- Fraud detection – 기존의 데이터에서 룰을 만든 후에 그 룰에 나타나지 않는 패턴으로 유저가 행동할 때 flag를 세팅함
  - 의사의 치료나 처방 – 과다하거나 불필요한 치료, 검사 또는 처방을 함
  - 의사들의 허위보험 청구 – overbilling (더 비싸게 청구), up-coding (10cm 꼬맨 것을 20cm로 청구)
  - 컴퓨터의 해킹탐지
  - 관공서에 사회보장제도로 의한 여러 가지 돈을 청구할 때에 돈이 여기저기로 세어나가는데 이런 것들을 찾아냄
- 시스템 failure 예측 – 네트웍의 정보를 보고 네트웍 failure를 예측하여 서버를 늘리거나 회선을 늘려서 문제가 없도록 함
- 서브그래프 마이닝
  - Weblog를 이용하여 navigation pattern 을 알아냄 – set으로 취급함
  - 화학구조를 분석함

Association Rules

- Given:
  - A database of customer transactions
  - Each transaction is a set of items
- Find all rules *X => Y* that correlate the presence of one set of items *X* with another set of items *Y*
  - Example: 98% of people who purchase diapers and baby food also buy beer.
  - Any number of items in the consequent/antecedent of a rule
  - Possible to specify constraints on rules (e.g., find only rules involving expensive imported products)

Support and Confidence

- $X \rightarrow Y$ [support, confidence]

$$\text{지지도(support)} = \frac{\#\text{ of transactions containing all the items in } X \cup Y}{\text{total }\#\text{ of transactions in the database}}$$

$$\text{신뢰도(confidence)} = \frac{\#\text{ of transactions that contain both } X \text{ and } Y}{\#\text{ of transactions containing } X}$$

- For minimum support (최소 지지도) = 50%, minimum confidence (최소 신뢰도) = 50%
  - B => C with 50% support and 66% confidence

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

Association Rule 찾는 방법

- 문제를 해결하기 위하여 2 개의 스텝으로 나누어 처리함
  - 스텝 1: Find all (frequent) itemsets that have minimum support
    - Most expensive phase
    - Lots of research
  - 스텝 2: Use the frequent itemsets to generate the desired rules
    - Generation is straight forward

Association Rule 찾는 방법

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

최소지지도=**40%**
최소신뢰도 = **100%**

- 스텝 1
  - 최소지지도 를 만족하는 frequent itemset들을 모두 찾음

Itemset	Sup
a	3
b	3
c	3
e	3
f	2

Itemset	Sup
a,c	2
a,f	2
b,c	2
b,e	3
c,e	2

Itemset	Sup
b,c,e	2

- 스텝 2
  - 모든 frequent itemset 으로부터 룰 생성
  - {b,c,e} 에서 아래 룰들을 다 만든 후에 신뢰도를 체크함
    - {b}->{c,e} (X)
    - {c}->{b,e}
    - {e}->{b,c}
    - {b,c}->{e} (O)
    - {b,e}->{c}
    - {c,e}->{b} (O)

Naïve Counting of All Itemsets

Itemsets & Counts

Itemset	Count
A	1
C	1
D	1
A,C	1
A,D	1
C,D	1
A,C,D	1

Transactions

TID	Items
10	A,C,D ☜
20	B,C,E
30	A,B,C,E
40	B,E

Naïve Counting of All Itemsets

Transactions

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

☜

Itemsets & Counts

Itemset	Count
A	1
C	2
D	1
A,C	1
A,D	1
C,D	1
A,C,D	1
B	1
E	1
B,C	1
B,E	1
C,E	1
B,C,E	1

Naïve Counting of All Itemsets

Transactions

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

Itemsets & Counts

Itemset	Count
A	2
C	3
D	1
A,C	2
A,D	1
C,D	1
A,C,D	1
B	2
E	2
B,C	2
B,E	2
C,E	2
B,C,E	2

Itemset	Count
A,B	1
A,E	1
A,B,C	1
A,B,E	1
A,B,C,E	1

Naïve Counting of All Itemsets

Transactions

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

Itemsets & Counts

Itemset	Count
A	2
C	3
D	1
A,C	2
A,D	1
C,D	1
A,C,D	1
B	3
E	3
B,C	2
B,E	3
C,E	2
B,C,E	2

Itemset	Count
A,B	1
A,E	1
A,B,C	1
A,B,E	1
A,B,C,E	1

Naïve Counting of All Itemsets

Transactions

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

$Sup_{min}=2$

Frequent itemsets

Itemset	Count
A	2
C	3
D	1
A,C	2
A,D	1
C,D	1
A,C,D	1
B	3
E	3
B,C	2
B,E	3
C,E	2
B,C,E	2

Itemset	Count
A,B	1
A,E	1
A,B,C	1
A,B,E	1
A,B,C,E	1

We may need 2^n itemset entries for counts !

Can we do better?

- 그냥 데이터를 보고 나오는 모든 상품들의 부분집합을 다 count 하면 exponential 한 개수의 부분집합을 count 하게 됨
- 모든 부분집합을 count 안 하는 방법이 있을까?
- Key Observation
  - Every subset of a frequent item set is also frequent item set.
  - If {beer, diaper, nuts} is frequent, {beer, diaper} must be frequent.
- If there is any item set which is infrequent, its superset will not be generated!
  - A powerful candidate set pruning technique.

Apriori: A Candidate Generation-and-Test Approach

- Apriori pruning principle: If there is any itemset which is infrequent, its superset should not be generated/tested! (Agrawal & Srikant @VLDB'94, Mannila, et al. @ KDD' 94)
- Method:
  - Initially, scan DB once to get frequent 1-itemset
  - Generate length (k+1) candidate itemsets from length k frequent itemsets
  - Test the candidates against DB
  - Terminate when no frequent or candidate set can b generated

Scalable Methods for Mining Frequent Patterns

- The downward closure property of frequent patterns
  - Any subset of a frequent itemset must be frequent
  - If **{beer, diaper, nuts}** is frequent, so is **{beer, diaper}**
  - i.e., every transaction having {beer, diaper, nuts} also contains {beer, diaper}
- Scalable mining methods: Three major approaches
  - Apriori (Agrawal & Srikant@VLDB'94)
  - Freq. pattern growth (FPgrowth—Han, Pei & Yin @SIGMOD'00)
  - Vertical data format approach (Charm—Zaki & Hsiao @SDM'02)

An Apriori Example

Min_sup=2

Data base D

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

Scan D

1-candidates

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

Freq 1-itemsets

Itemset	Sup
a	3
b	3
c	3
e	3
f	2

2-candidates

Itemset
ab
ac
ae
af
bc
be
bf
ce
cf
ef

Scan D

Itemset	Sup
ab	1
ac	2
ae	1
af	2
bc	2
be	3
bf	0
ce	2
cf	0
ef	0

Freq 2-itemsets

Itemset	Sup
ac	2
af	2
bc	2
be	3
ce	2

3-candidates

Itemset
acf
bce

pruning

3-candidates

Itemset
bce

Scan D

Itemset	Sup
bce	2

Freq 3-itemsets

Itemset	Sup
bce	2

- From [Agrawal, Srikant 1994]

The Apriori Algorithm

- C_k: Candidate itemset of size k
- F_k : frequent itemset of size k

- F_1 = {frequent items};
- for (k = 1; F_k !=$\emptyset$; k++) do
  - C_{k+1} = candidates generated from F_k;
  - for each transaction t in database do increment the count of all candidates in C_{k+1} that are contained in t
  - F_{k+1} = candidates in C_{k+1} with min_support
- return $\cup_k F_k$;

Discovering Rules

Naïve Algorithm:

for each frequent itemset f **do**
    for each subset c of f **do**
        if (support(f)/support(f-c) ≥ minconf) **then**
            output the rule (f-c)➔ c,
                with confidence = support(f)/support(f-c)
                and support = support(f)

Discovering Rules

- Consider the rule (f-c)➔c
- Now, if c1 is a subset of c
  - f-c1 is a superset of C

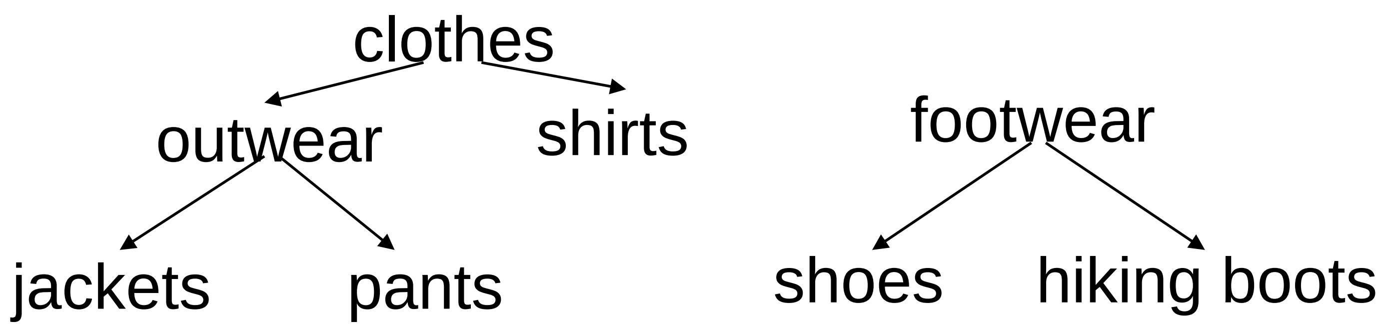
    support(f-c1) ≤ support(f-c)

    support(f)/support(f-c1) ≥ support(f)/support(f-c)

    conf((f-c1)➔c1) ≥ conf((f-c)➔c)
- So, if a consequent c generates a valid rule, so do all subsets of c
- Can use the apriori candidate generation algorithm to limit number of possible rules tested.
- Consider a frequent itemset ABCDE
  - If ACDE➔B and ABCE➔D are the only one-consequent rules with minimum confidence, then ACE ➔ BD is the only other rule that needs to be ested.

Generalized Association Rules

- Hierarchies over items (e.g. UPC codes)



- Associations across hierarchies:
  - The rule clothes => footwear may hold even if clothes => shoes do not hold
- [Srikant, Agrawal 95]
- [Han, Fu 95]

Quantitative Association Rules

- [Srikant, Agrawal 96]

RecordID	Age	Married	NumCars
100	23	No	1
200	25	Yes	1
300	29	No	0
400	34	Yes	2
500	38	Yes	2

- **Quantitative attributes (e.g.age,income)**
- **Categorical attributes (e.g.make of car)**

min support = 40% min confidence = 50%

Sample Rules	Support	Confidence
<age:30..39> and <married: yes> ==> <numCars:2>	40%	100%
<NumCars: 0..1> ==> <Married: No>	40%	66.70%

from [Srikant, Agrawal 96]

Temporal Association Rules

- 데이터에 있는 시간 정보 까지 이용함
- Example:
  - {diaper} -> {beer} (support = 5%, confidence = 87%)
  - 이 룰의 지지도가 평일 6 시에서 9 PM 까지 시간에는 25%로 점프함
- Problem: How to find rules that follow interesting user-defined temporal patterns
- 각각의 시간마다 모든 룰을 일단 다 찾아 내는 것보다 더 효율적인 알고리즘을 만들어 내는 것이 Challenge임
- [Ozden, Ramaswamy, Silberschatz 98]
- [Ramaswamy, Mahajan, Silberschatz 98]

FP-Tree Algorithm

- Jiawei Han, Jian Pei, Yiwen Yin: *Mining Frequent Patterns without Candidate Generation In* ACM SIGMOD 2000

Why FP-Tree and not Apriori?

- Apriori works well except when:
  - Lots of frequent patterns
    - Big set of items
  - Low minimum support threshold
    - Long patterns
- Why: Candidate sets become huge
  - Discovering pattern of length 100 requires at least 2^{100}candidates (number of subsets)
  - Repeated database scans costly (long patterns)
- Multiple database scans are costly
- Bottleneck: candidate-generation-and-test
- Can we avoid candidate generation?

FP-Tree: Ideas

- Avoid candidate set explosion by:
  - Compact tree data structure
    - Avoid repeated database scans
  - Restricted test-only
    - Apriori: restricted generation-and-test
  - Search divide-and-conquer based
    - Apriori: breadth-first

Mining Frequent Patterns Without Candidate Generation

- Grow long patterns from short ones using local frequent items
  - "abc" is a frequent pattern
  - Get all transactions having "abc": DB|abc
  - "d" is a local frequent item in DB|abc → abcd is a frequent pattern

Construct FP-tree from a Transaction Database

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}*

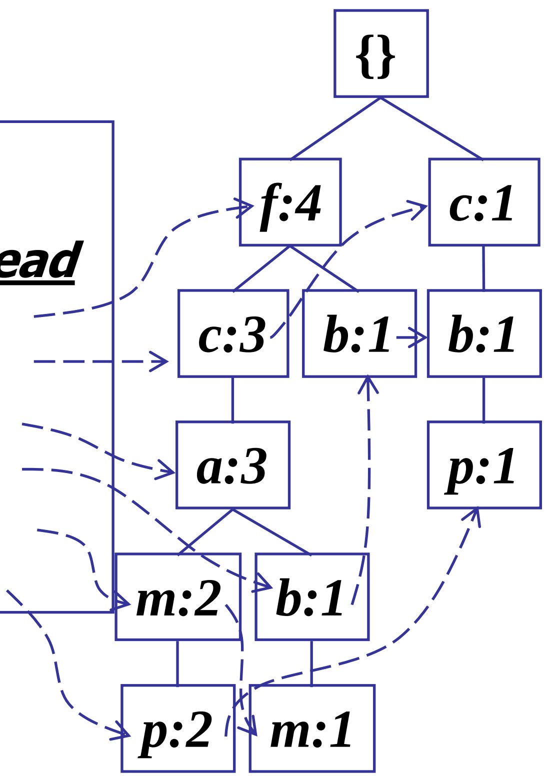
min_support = 3

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

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

Header Table

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



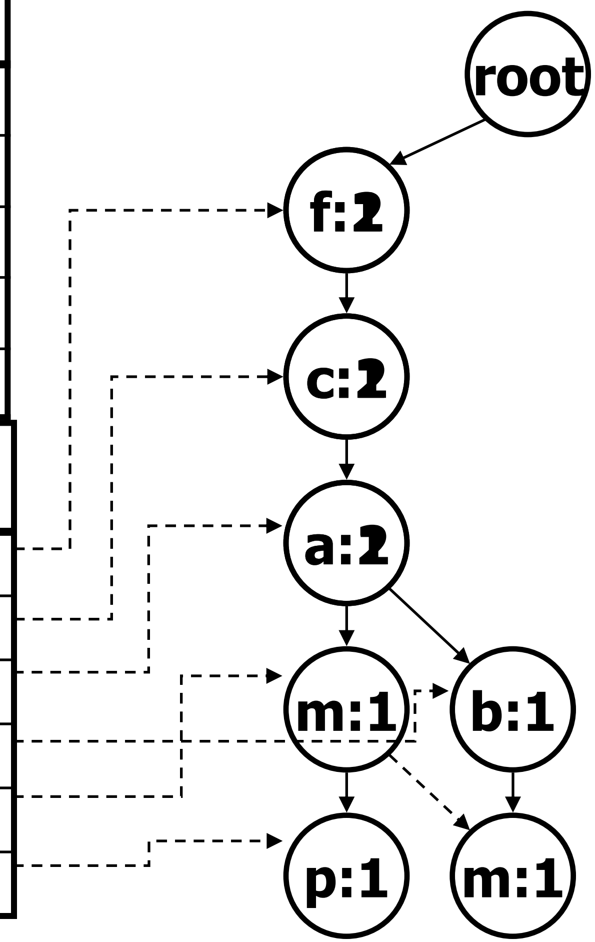
Example : Construction of FP-tree

Min_sup = 3

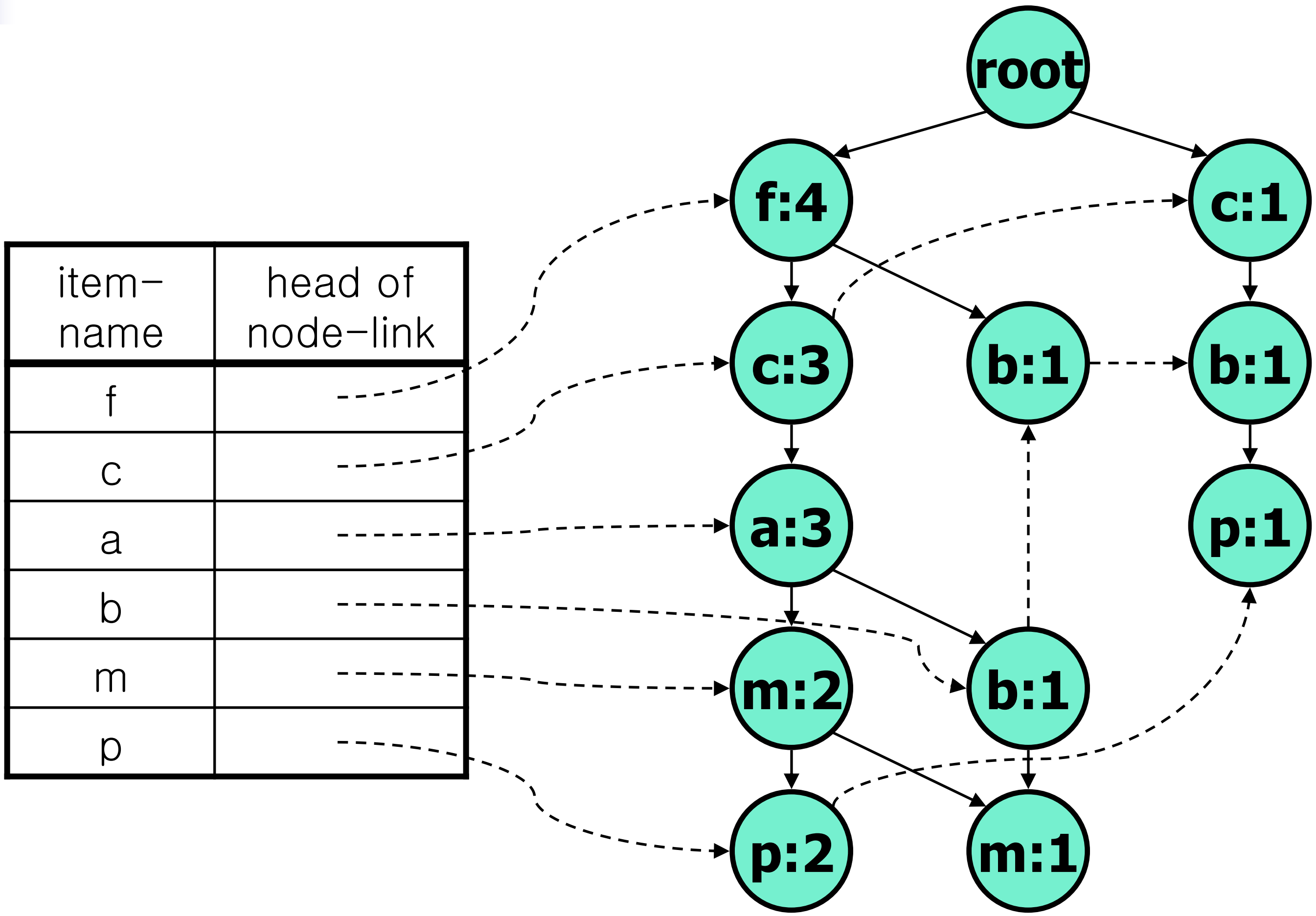
TID	Items bought	(Ordered) frequent items
100	f, a, c, d, g, i, m, p	f, c, a, m, p
200	a, b, c, f, l, m, o	f, c, a, b, m
300	b, f, h, j, o	f, b
400	b, c, k, s, p	c, b, p
500	a, f, c, e, l, p, m, n	f, c, a, m, p

item	support
f	4
c	4
a	3
b	3
m	3
p	3

item−name	head of node−link
f	
c	
a	
b	
m	
p	



An Example of a complete FP-tree



Properties of FP-tree

- FP-tree contains the complete information of DB relevant to frequent pattern mining (completeness)

- A lot of sharing of frequent items makes the FP-tree more compact (compactness)
  - e.g. in MaxMiner experiment
    - The total # of occurrence of frequent items : 2,219,609
    - The total # of nodes in the FP-tree : 13,339

FP-Tree: Properties

- The FP-Tree contains everything from the database we need to know for mining frequent patterns
- The size of the FP-tree is ≤ Occurrence of frequent patterns in database

Properties of FP-tree

- FP-tree contains the complete information of DB relevant to frequent pattern mining (completeness)

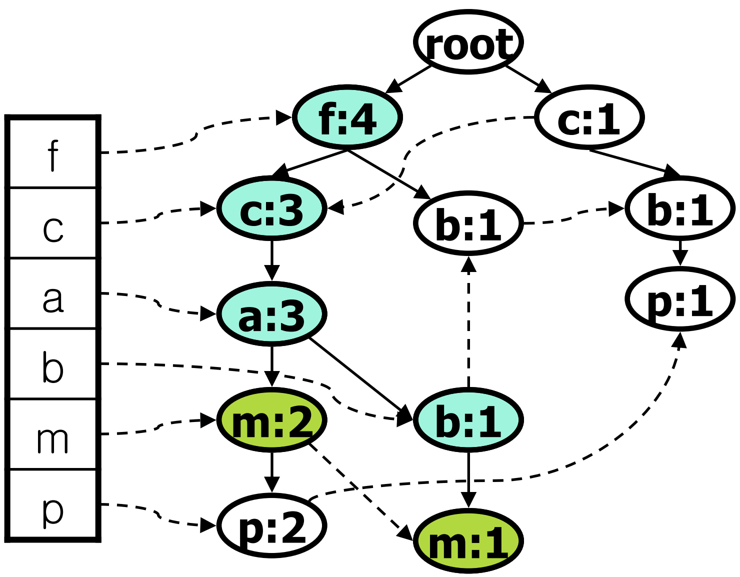
- A lot of sharing of frequent items makes the FP-tree more compact (compactness)
  - e.g. in MaxMiner experiment
    - The total # of occurrence of frequent items : 2,219,609
    - The total # of nodes in the FP-tree : 13,339

Mining Frequent Patterns

- How do we get find all frequent patterns from the FP-Tree?
  - Intuitively:
    1)Find all frequent patterns containing one of the items
    2)Then find all frequent patterns containing the next item but NOT containing the previous one
    3)Repeat 2) until we're out of items

Conditional Pattern Base

- A sub-pattern base under the condition of existence of a certain pattern
- Example (min_sup = 3)



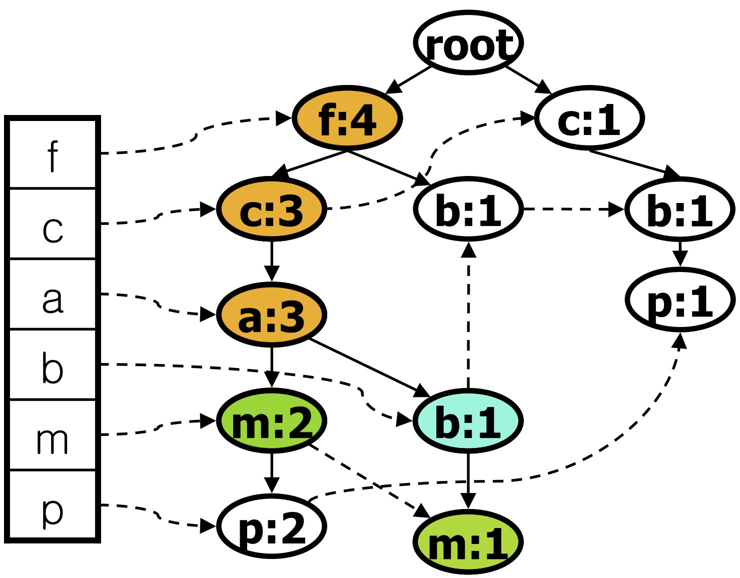
Nodes that contribute m's cond. pattern bases

m's cond. pattern bases

- (fca:2), (fcab:1)

Conditional FP-tree

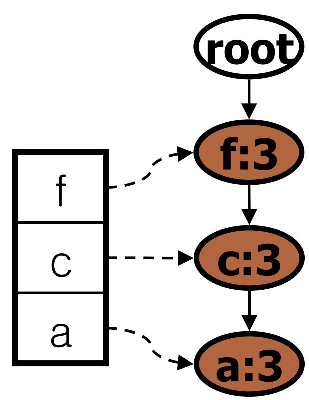
- FP-tree on the conditional pattern bases of a certain item
- If FP-tree consists of single path, all the combinations of items in the path are the freq patterns
- Example (min_sup = 3)



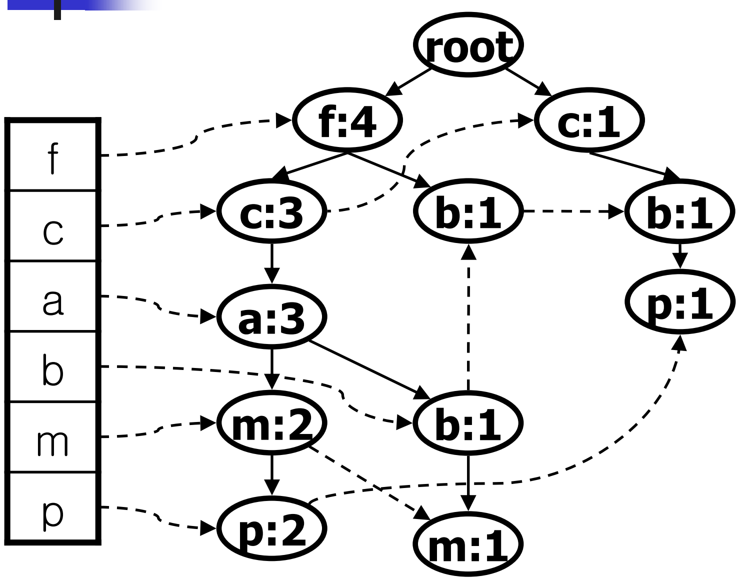
m's cond. pattern bases
- (fca:2), (fcab:1)

The nodes that contribute to m's cond. FP-tree

m's cond. FP-tree
- (fca:3)



Example : FP-growth



Min_sup=3

item	conditional pattern base	conditional FP−tree	result freq. pattern
p	(fcam:2), (cb:1)	(c:3)	p, cp
m	(fca:2), (fcab:1)	(fca:3)	m, am, cm, fm, cam, fam, fcm, fcam
b	(fca:1), (f:1). (c:1)	none	b
a	(fc:3)	(fc:3)	a, ca, fa, fca
c	(f:3)	(f:3)	c, fc
f	none	none	f

Benefits of the FP-tree Structure

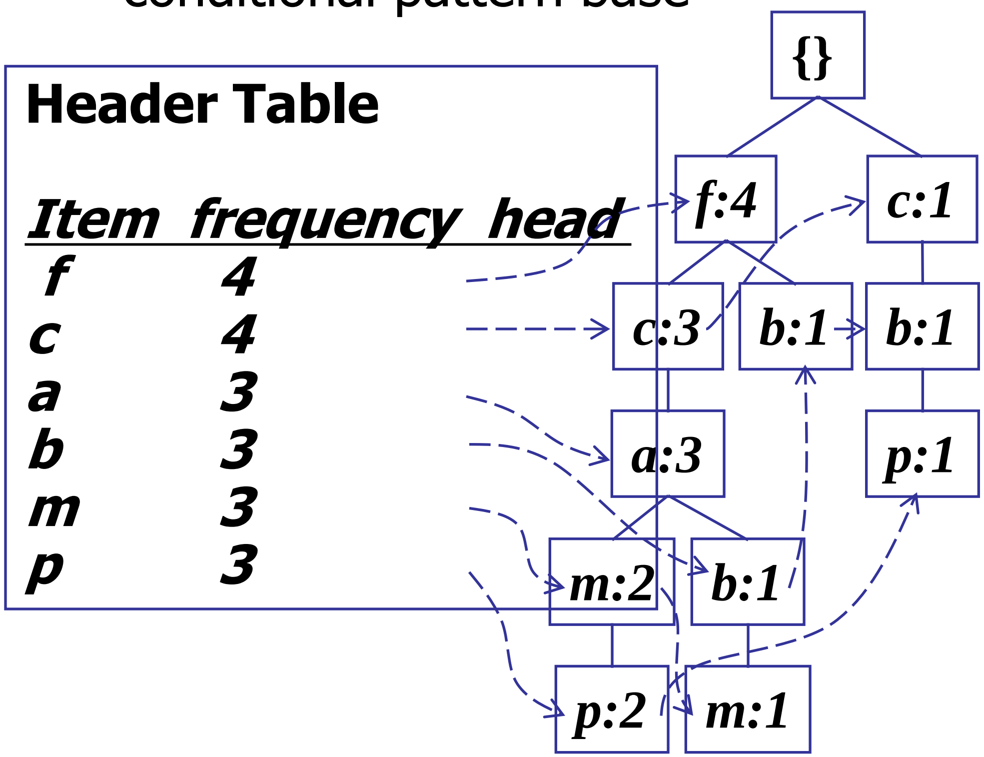
- Completeness
  - Preserve complete information for frequent pattern mining
  - Never break a long pattern of any transaction
- Compactness
  - Reduce irrelevant info—infrequent items are gone
  - Items in frequency descending order: the more frequently occurring, the more likely to be shared
  - Never be larger than the original database (not count node-links and the *count* field)
  - For Connect-4 DB, compression ratio could be over 100

Partition Patterns and Databases

- Frequent patterns can be partitioned into subsets according to f-list
  - F-list=f-c-a-b-m-p
  - Patterns containing p
  - Patterns having m but no p
  - …
  - Patterns having c but no a nor b, m, p
  - Pattern f
- Completeness and non-redundency

Find Patterns Having P From P-conditional Database

- 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 *transformed prefix paths* of item *p* to form *p*'s conditional pattern base

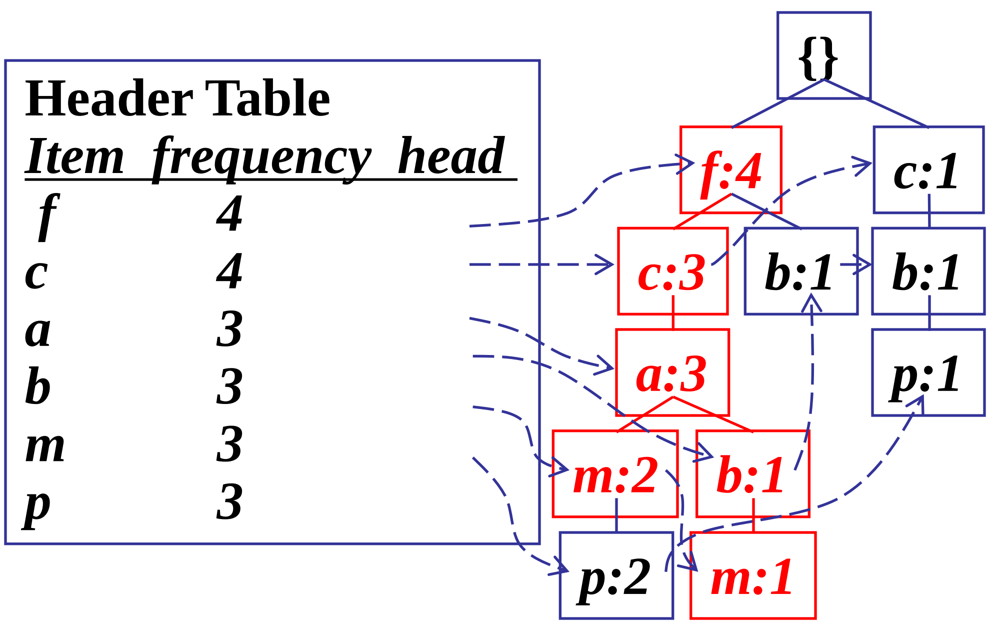


Conditional **pattern bases**

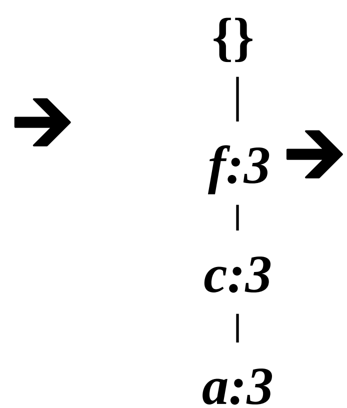
item	*cond. pattern base*
c	*f:3*
a	*fc:3*
b	*fca:1, f:1, c:1*
m	*fca:2, fcab:1*
p	*fcam:2, cb:1*

From Conditional Pattern-bases to Conditional FP-trees

- 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



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

Recursion: Mining Each Conditional FP-tree

{}
|
f:3
|
c:3
|
a:3

m-conditional FP-tree

Cond. pattern base of "am": (fc:3)

{}
|
f:3
|
c:3

am-conditional FP-tree

Cond. pattern base of "cm": (f:3)

{}
|
f:3

cm-conditional FP-tree

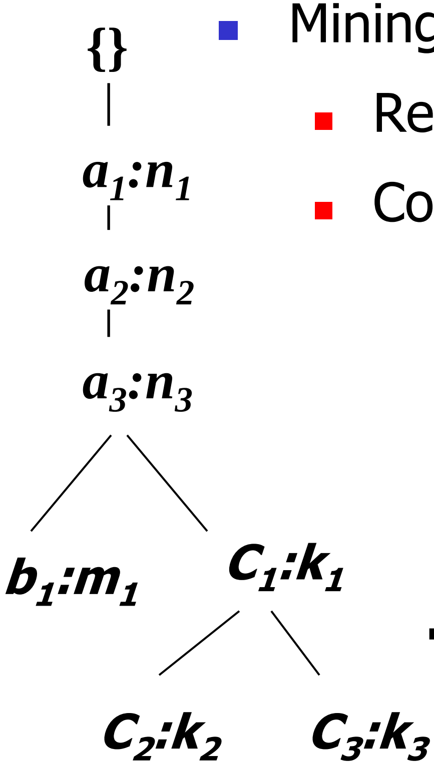
Cond. pattern base of "cam": (f:3)

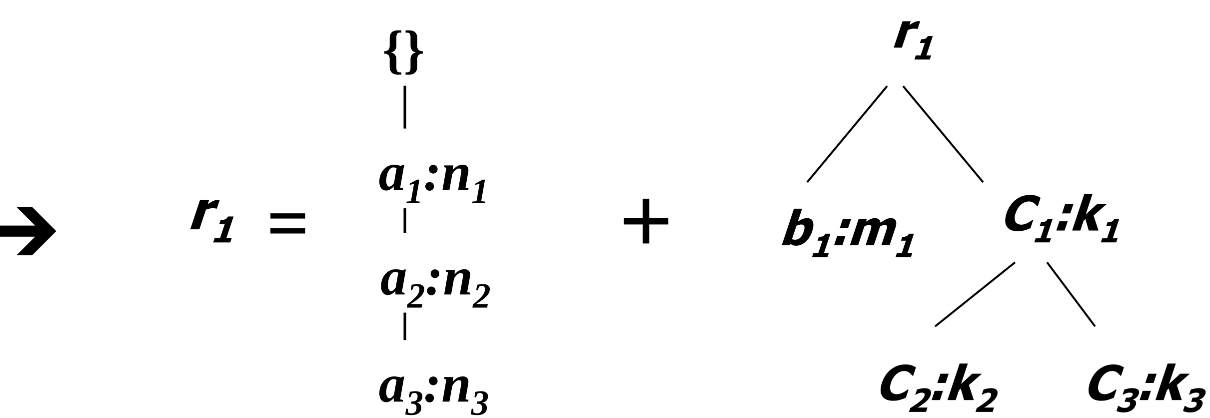
{}
|
f:3

cam-conditional FP-tree

A Special Case: Single Prefix Path in FP-tree

- Suppose a (conditional) FP-tree T has a shared single prefix-path P
- Mining can be decomposed into two parts
  - Reduction of the single prefix path into one node
  - Concatenation of the mining results of the two parts





Mining Frequent Patterns With FP-trees

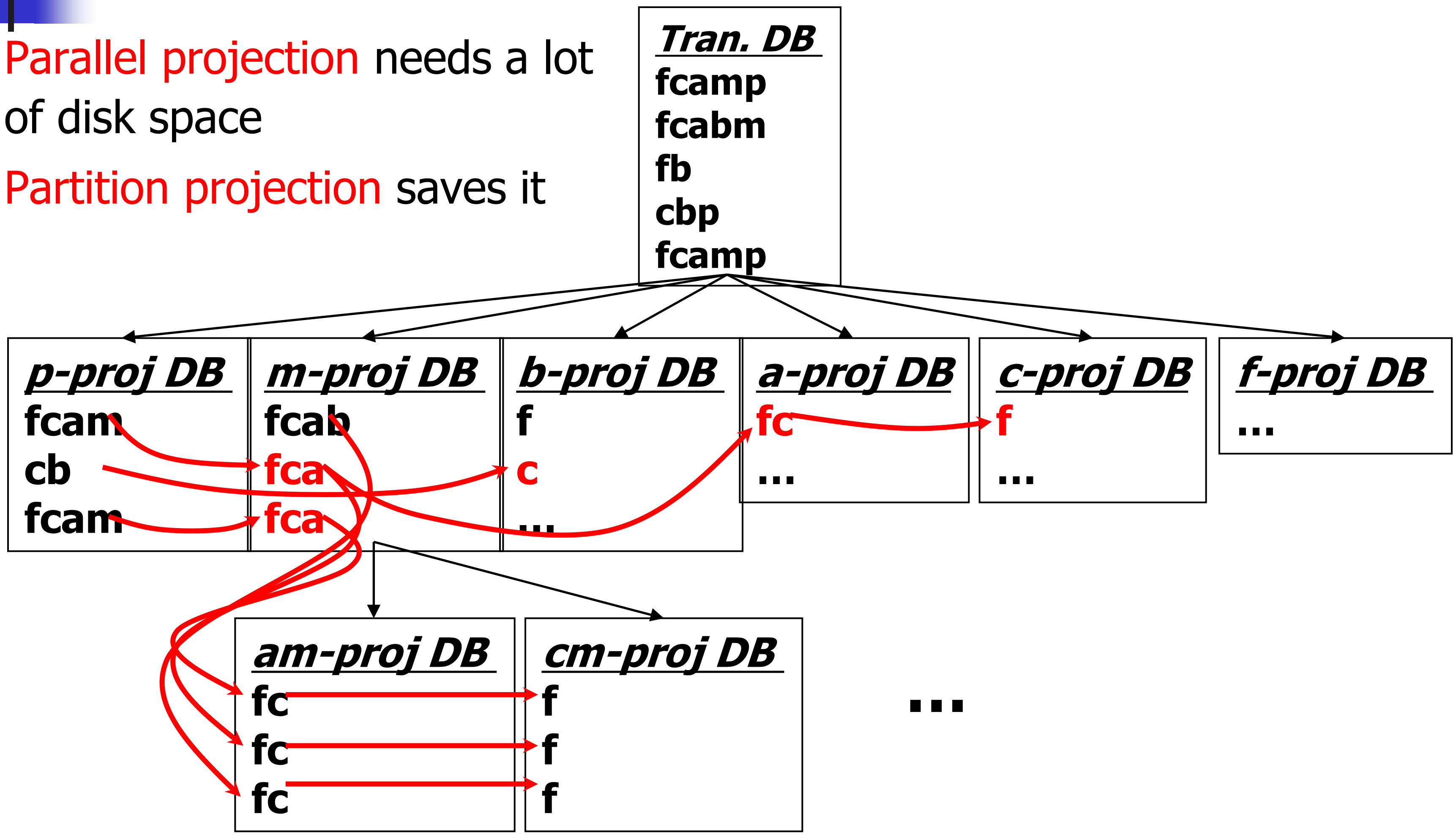
- Idea: Frequent pattern growth
  - Recursively grow frequent patterns by pattern and database partition
- Method
  - For each frequent item, construct its conditional pattern-base, and then its conditional FP-tree
  - Repeat the process on each newly created conditional FP-tree
  - Until the resulting FP-tree is empty, or it contains only one path—single path will generate all the combinations of its sub-paths, each of which is a frequent pattern

Scaling FP-growth by DB Projection

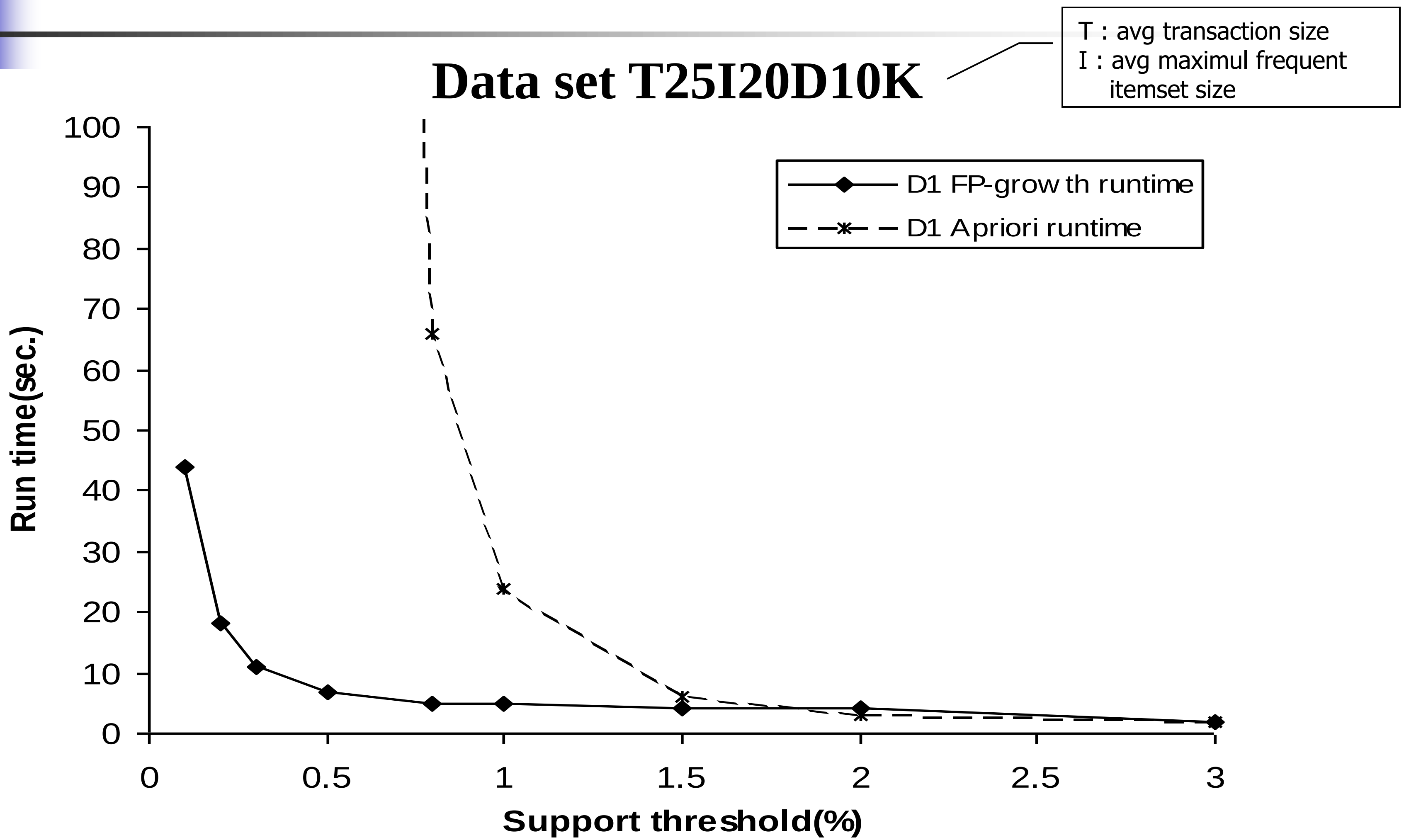
- FP-tree cannot fit in memory?—DB projection
- First partition a database into a set of projected DBs
- Then construct and mine FP-tree for each projected DB
- Parallel projection vs. Partition projection techniques
  - Parallel projection is space costly

Partition-based Projection

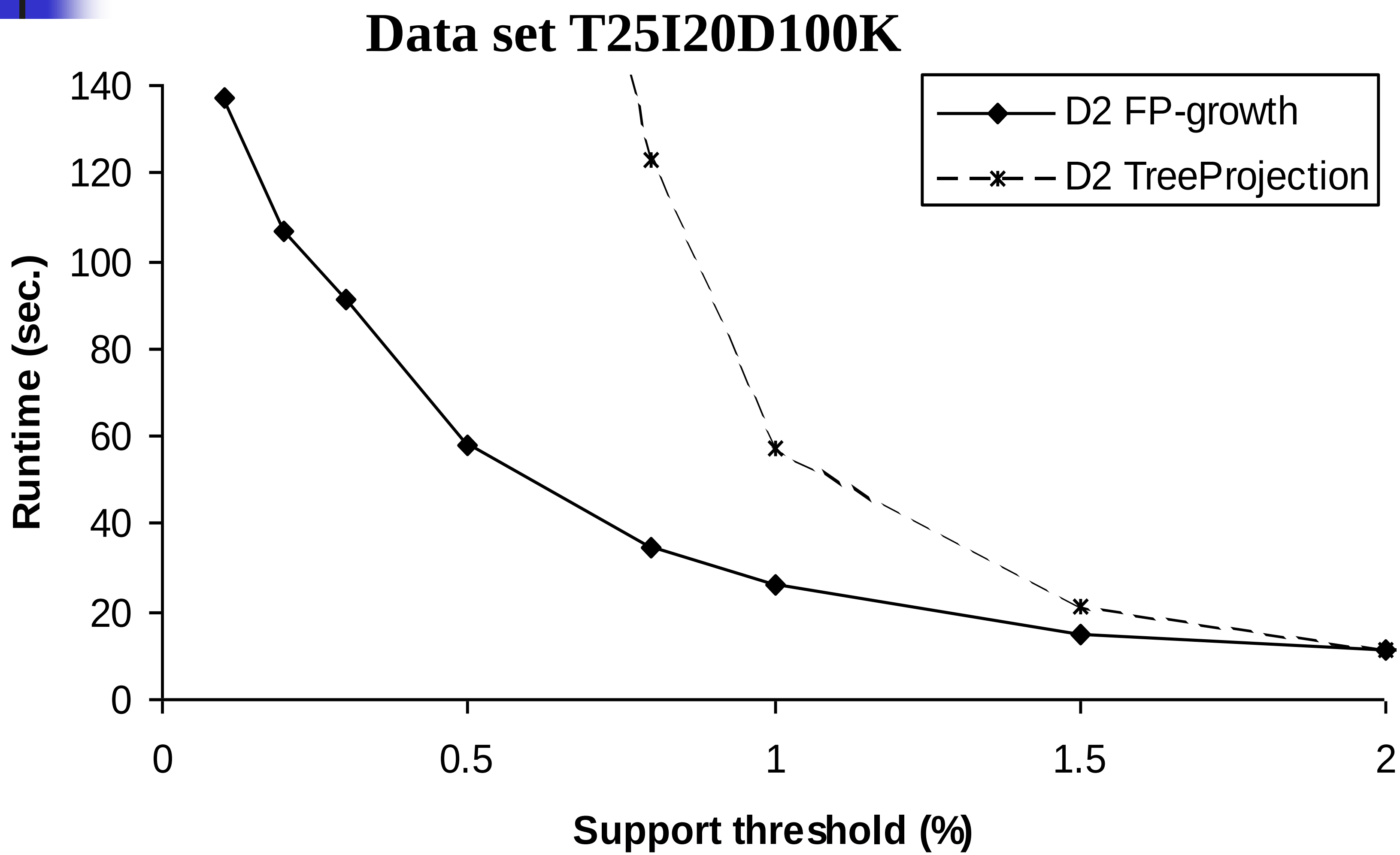
- Parallel projection needs a lot of disk space
- Partition projection saves it



FP-Growth vs. Apriori: Scalability With the Support Threshold



FP-Growth vs. Tree-Projection: Scalability with the Support Threshold



Why Is FP-Growth the Winner?

- Divide-and-conquer:
  - Decompose both the mining task and DB according to the frequent patterns obtained so far
  - Leads to focused search of smaller databases
- Other factors
  - No candidate generation, no candidate test
  - Compressed database: FP-tree structure
  - No repeated scan of entire database
  - Basic ops—counting local freq items and building sub FP-tree, no pattern search and matching

Implications of the Methodology

- Mining closed frequent itemsets and max-patterns
  - CLOSET (DMKD'00)
- Mining sequential patterns
  - FreeSpan (KDD'00), PrefixSpan (ICDE'01)
- Constraint-based mining of frequent patterns
  - Convertible constraints (KDD'00, ICDE'01)
- Computing iceberg data cubes with complex measures
  - H-tree and H-cubing algorithm (SIGMOD'01)

MaxMiner: Mining Max-patterns

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

- 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, CDE, DE,

Potential max-patterns

- Since BCDE is a max-pattern, no need to check BCD, BDE, CDE in later scan
- R. Bayardo. Efficiently mining long patterns from databases. In *SIGMOD'98*

Mining Frequent Closed Patterns: CLOSET

- Flist: list of all frequent items in support ascending 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
- J. Pei, J. Han & R. Mao. CLOSET: An Efficient Algorithm for Mining Frequent Closed Itemsets", DMKD'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

CLOSET+: Mining Closed Itemsets by Pattern-Growth

- Itemset merging: if Y appears in every occurrence of X, then Y is merged with X
- Sub-itemset pruning: if Y ⊃ X, and sup(X) = sup(Y), X and all of X's descendants in the set enumeration tree can be pruned
- Hybrid tree projection
  - Bottom-up physical tree-projection
  - Top-down pseudo tree-projection
- Item skipping: if a local frequent item has the same support in several header tables at different levels, one can prune it from the header table at higher levels
- Efficient subset checking

CHARM: Mining by Exploring Vertical Data Format

- Vertical format: $t(AB) = \{T_{11}, T_{25}, \ldots\}$
  - tid-list: list of trans.-ids containing an itemset
- Deriving closed 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
- Using diffset to accelerate mining
  - Only keep track of differences of tids
  - $t(X) = \{T_1, T_2, T_3\}$, $t(XY) = \{T_1, T_3\}$
  - Diffset $(XY, X) = \{T_2\}$
- Eclat/MaxEclat (Zaki et al. @KDD'97), VIPER(P. Shenoy et al.@SIGMOD'00), CHARM (Zaki & Hsiao@SDM'02)

Further Improvements of Mining Methods

- AFOPT (Liu, et al. @ KDD'03)
  - A "push-right" method for mining condensed frequent pattern (CFP) tree
- Carpenter (Pan, et al. @ KDD'03)
  - Mine data sets with small rows but numerous columns
  - Construct a row-enumeration tree for efficient mining