Data Mining:

Concepts and Techniques

(3rd ed.)

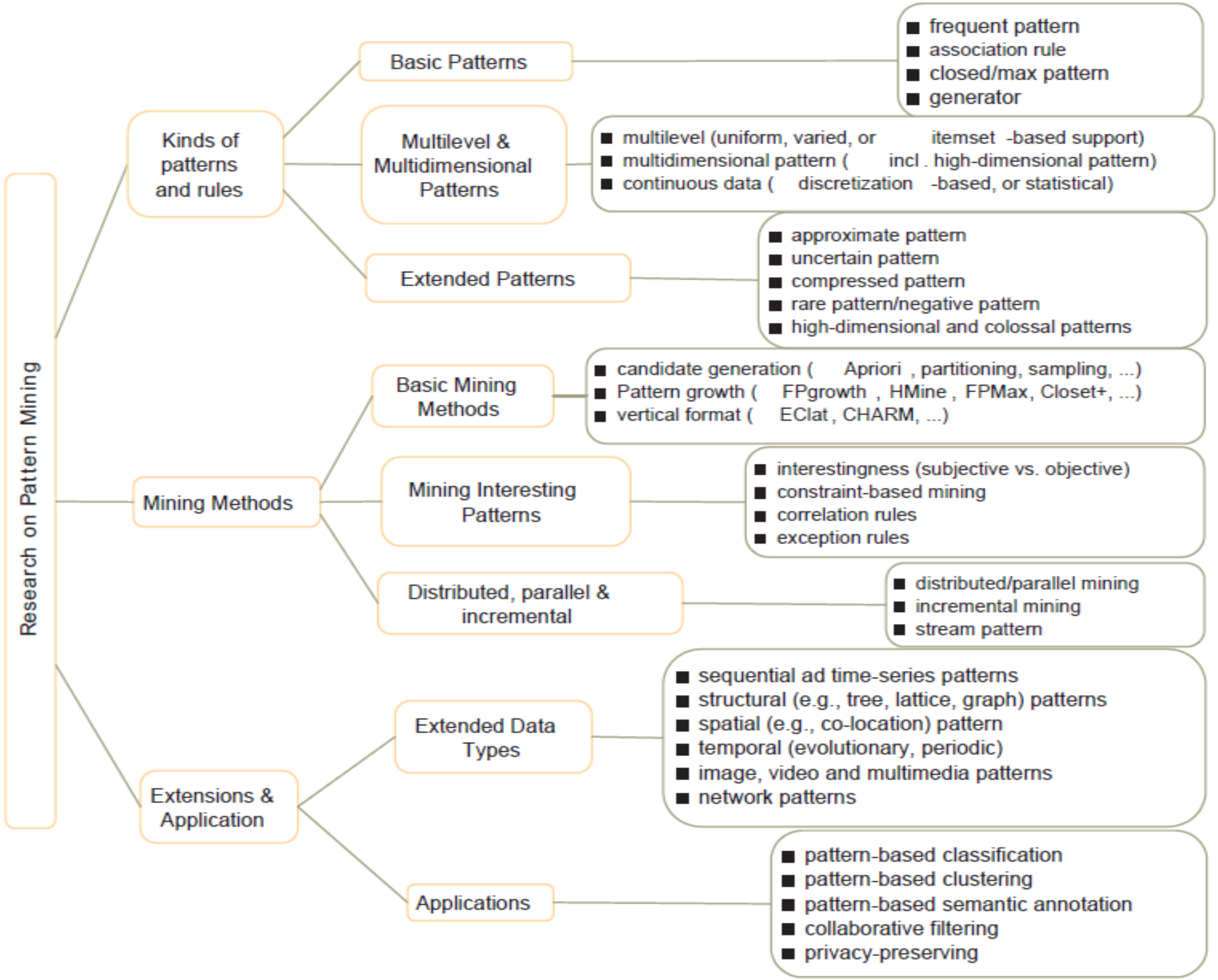
— Chapter 7 —

Slides Courtesy of Textbook

Chapter 7 : Advanced Frequent Pattern Mining

- Pattern Mining: A Road Map
- ~~Pattern Mining in Multi-Level, Multi-Dimensional Space~~
- Constraint-Based Frequent Pattern Mining
- ~~Mining High-Dimensional Data and Colossal Patterns~~
- ~~Mining Compressed or Approximate Patterns~~
- Sequential Pattern Mining
- Graph Pattern Mining
- Summary

Research on Pattern Mining: A Road Map



Chapter 7 : Advanced Frequent Pattern Mining

- Pattern Mining: A Road Map
- ~~Pattern Mining in Multi-Level, Multi-Dimensional Space~~
- Constraint-Based Frequent Pattern Mining
- ~~Mining High-Dimensional Data and Colossal Patterns~~
- ~~Mining Compressed or Approximate Patterns~~
- Sequential Pattern Mining
- Graph Pattern Mining
- Summary

Constraint-based (Query-Directed) Mining

- Finding all the patterns in a database autonomously? — unrealistic!
  - The patterns could be too many but not focused!
- Data mining should be an interactive process
  - User directs what to be mined using a data mining query language (or a graphical user interface)
- Constraint-based mining
  - User flexibility: provides constraints on what to be mined
  - Optimization: explores such constraints for efficient mining — constraint-based mining: constraint-pushing, similar to push selection first in DB query processing
  - Note: still find all the answers satisfying constraints, not finding some answers in "heuristic search"

Constraints in Data Mining

- Rule (or pattern) constraint
  - small sales (price < $10) triggers big sales (sum > $200)
- Other Types of Constraints:
  - Knowledge type constraint:
    - classification, association, etc.
  - Data constraint — using SQL-like queries
    - find product pairs sold together in stores in Chicago this year
  - Dimension/level constraint
    - in relevance to region, price, brand, customer category
  - Interestingness constraint
    - strong rules: min_support ≥ 3%, min_confidence ≥ 60%

Constraint-Based Frequent Pattern Mining

- **Pattern space pruning** constraints
  - Anti-monotonic: If constraint c is violated, its further mining can be terminated
  - Monotonic: If c is satisfied, no need to check c again
  - Succinct: c must be satisfied, so one can start with the data sets satisfying c
  - Convertible: c is not monotonic nor anti-monotonic, but it can be converted into it if items in the transaction can be properly ordered

Pattern Space Pruning with Anti-Monotonicity Constraints

- *pattern anti-monotone*: For a constraint C, if the super pattern satisfies C, all of its sub-patterns do so too
- *data anti-monotone:* If an itemset S **violates** the constraint, so does any of its superset
- Ex. 1. $sum(S.price) \leq v$ is anti-monotonic
- Ex. 2. max(S.profit) ≤ 15 is anti-monotonic
  - Itemset *ab* violates C
  - So does every superset of *ab*
- Ex. 3. $sum(S.Price) \geq v$ is not anti-monotonic
- Ex. 4. *support count* is anti-monotone: core property used in Apriori

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	Price
a	40	10
b	0	5
c	-20	20
d	10	10
e	-30	100
f	30	40
g	20	50
h	-10	60

Pattern Space Pruning with Monotonicity Constraints

- A constraint C is *monotonic* if the pattern satisfies C, we do not need to check C in subsequent mining
- Ex. 1. *sum(S.Price)* ≥ *v* is monotonic
- Ex. 2. *min(S.Price)* ≤ *v* is monotonic
- Ex. 3. C: max(S.profit) ≥ 15
  - Itemset *ab* satisfies C
  - So does every superset of *ab*

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	Price
a	40	10
b	0	5
c	-20	20
d	10	10
e	-30	100
f	30	40
g	20	50
h	-10	60

Pattern Space Pruning with Succinctness

- Succinctness:
  - Exists a set A_1 such that for any set S satisfying C , S and A1 have a non-empty intersection, i.e., to satisfy C, S contains a subset belonging to A_1
  - $min(S.Price) \leq v$ is succinct
  - $sum(S.Price) \geq v$ is not succinct
- Optimization: If C is succinct, C is pre-counting pushable

Apriori + Constraint

Database D

TID	Items
100	1 3 4
200	2 3 5
300	1 2 3 5
400	2 5

Scan D

C_1

itemset	sup.
{1}	2
{2}	3
{3}	3
{4}	1
{5}	3

L_1

itemset	sup.
{1}	2
{2}	3
{3}	3
~~{5}~~	~~3~~

C_2

itemset
{1 2}
{1 3}
~~{1 5}~~
~~{2 3}~~
~~{2 5}~~
~~{3 5}~~

Scan D

C_2

itemset	sup
{1 2}	1
{1 3}	2
~~{1 5}~~	~~1~~
{2 3}	2
~~{2 5}~~	~~3~~
~~{3 5}~~	~~2~~

L_2

itemset	sup
{1 3}	2
~~{2 3}~~	~~2~~
~~{2 5}~~	~~3~~
~~{3 5}~~	~~2~~

C_3

itemset
~~{2 3 5}~~

Scan D

L_3

itemset	sup
~~{2 3 5}~~	~~2~~

Constraint:

Sum{S.price} < 5

Constrained Apriori : Push a Succinct Constraint Deep

Database D

TID	Items
100	1 3 4
200	2 3 5
300	1 2 3 5
400	2 5

Scan D

C_1

itemset	sup.
{1}	2
{2}	3
{3}	3
{4}	1
{5}	3

L_1

itemset	sup.
{1}	2
{2}	3
{3}	3
{5}	3

C_2

itemset
{1 2}
{1 3}
{1 5}
~~{2 3}~~
~~{2 5}~~
~~{3 5}~~

not immediately to be used

Scan D

C_2

itemset	sup
{1 2}	1
{1 3}	2
{1 5}	1
~~{2 3}~~	~~2~~
~~{2 5}~~	~~3~~
~~{3 5}~~	~~2~~

L_2

itemset	sup
{1 3}	2
~~{2 3}~~	~~2~~
~~{2 5}~~	~~3~~
~~{3 5}~~	~~2~~

C_3

itemset
~~{2 3 5}~~

Scan D

L_3

itemset	sup
~~{2 3 5}~~	~~2~~

Constraint:

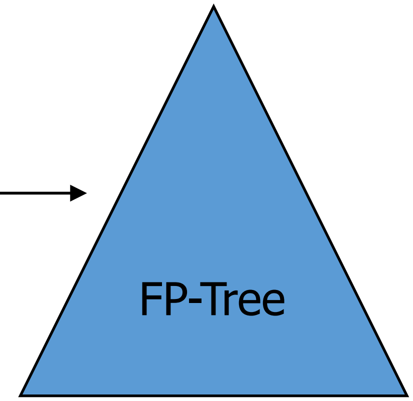
min{S.price } <= 1

Constrained FP-Growth: Push a Succinct Constraint Deep

TID	Items
100	1 3 4
200	2 3 5
300	1 2 3 5
400	2 5

Remove infrequent length 1

TID	Items
100	1 3
200	2 3 5
300	1 2 3 5
400	2 5



1-Projected DB

TID	Items
100	3 4
300	2 3 5

No Need to project on 2, 3, or 5

Constraint:

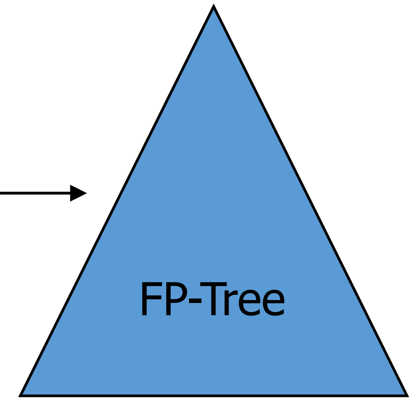
min{S.price } <= 1

Constrained FP-Growth: Push a Data Anti-monotonic Constraint Deep

Remove from data

TID	Items
100	1 3 4
~~200~~	~~2 3 5~~
300	1 2 3 5
~~400~~	~~2 5~~

TID	Items
100	1 3
300	1 3



Single branch, we are done

Constraint:

min{S.price } <= 1

Convertible Constraints: Ordering Data in Transactions

- Convertible constraints are converted into anti-monotone or monotone by properly ordering items
- Examine C: avg(*S*.profit) ≥ 25
  - Order items in value-descending order
    - *<a, f, g, d, b, h, c, e>*
  - If an itemset *afb* violates C
    - So does *afbh, afb**
    - It becomes anti-monotone!

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

Strongly Convertible Constraints

- Strongly convertible constraints can be converted to both monotone or anti-monotone constraints.
- avg(X) ≥ 25 is convertible anti-monotone
  - item value descending order R:
    - *<a, f, g, d, b, h, c, e>*
  - If an itemset *af* violates a constraint C, so does every itemset with *af* as prefix, such as *afd*
- avg(X) ≥ 25 is convertible monotone
  - item value ascending order R^{-1}:
    - *<e, c, h, b, d, g, f, a>*
  - If an itemset *d* satisfies a constraint *C*, so does itemsets *df* and *dfa*, which having *d* as a prefix
- Thus, avg(X) ≥ 25 is strongly convertible

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

Can Apriori Handle Convertible Constraints?

- A convertible (not monotone nor anti-monotone nor succinct) constraint cannot be pushed deep into the an Apriori mining algorithm
  - Within the level wise framework, no direct pruning based on the constraint can be made
  - Itemset df violates constraint C: avg(X) >= 25
  - Since adf satisfies C, Apriori needs df to assemble adf, df cannot be pruned
- But it can be pushed into frequent-pattern growth framework!

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

Pattern Space Pruning w. Convertible Constraints

- C: avg(X) >= 25, min_sup=2
- List items in every transaction in value descending order R: <a, f, g, d, b, h, c, e>
  - C is convertible anti-monotone w.r.t. R
- Scan TDB once
  - remove infrequent items
    - Item h is dropped
  - Itemsets a and f are good, …
- Projection-based mining
  - Imposing an appropriate order on item projection
  - Many tough constraints can be converted into (anti)-monotone

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

TDB (min_sup=2)

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

Handling Multiple Constraints

- Different constraints may require different or even conflicting item-ordering
- If there exists an order *R* s.t. both C_1 and C_2 are convertible w.r.t. *R,* then there is no conflict between the two convertible constraints
- If there exists conflict on order of items
  - Try to satisfy one constraint first
  - Then using the order for the other constraint to mine frequent itemsets in the corresponding projected database

Constraint-Based Mining — A General Picture

Constraint	Anti-monotone	Monotone	Succinct
$v \in S$	no	yes	yes
$S \supseteq V$	no	yes	yes
$S \subseteq V$	yes	no	yes
$min(S) \le v$	no	yes	yes
$min(S) \ge v$	yes	no	yes
$max(S) \le v$	yes	no	yes
$max(S) \ge v$	no	yes	yes
$count(S) \le v$	yes	no	weakly
$count(S) \ge v$	no	yes	weakly
$sum(S) \le v$ ($a \in S, a \ge 0$)	yes	no	no
$sum(S) \ge v$ ($a \in S, a \ge 0$)	no	yes	no
$range(S) \le v$	yes	no	no
$range(S) \ge v$	no	yes	no
$avg(S)\ \theta\ v, \theta \in \{ =, \le, \ge \}$	convertible	convertible	no
$support(S) \ge \xi$	yes	no	no
$support(S) \le \xi$	no	yes	no

Chapter 7 : Advanced Frequent Pattern Mining

- Pattern Mining: A Road Map
- ~~Pattern Mining in Multi-Level, Multi-Dimensional Space~~
- Constraint-Based Frequent Pattern Mining
- ~~Mining High-Dimensional Data and Colossal Patterns~~
- ~~Mining Compressed or Approximate Patterns~~
- Sequential Pattern Mining
- Graph Pattern Mining
- Summary

Sequence Databases & Sequential Patterns

- Transaction databases, time-series databases vs. sequence databases
- Frequent patterns vs. (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 treatments, natural disasters (e.g., earthquakes), science & eng. processes, stocks and markets, etc.
  - Telephone calling patterns, Weblog click streams
  - Program execution sequence data sets
  - DNA sequences and gene structures

What Is Sequential Pattern Mining?

- Given a set of sequences (ordered lists of itemsets), 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*

Sequential pattern mining: find the complete set of patterns, satisfying the minimum support (frequency) threshold

Sequential Pattern Mining Algorithms

- Concept introduction and an initial Apriori-like algorithm
  - Agrawal & Srikant: Mining sequential patterns, ICDE'95
- Requirement: efficient, scalable, complete, minimal database scans, and be able to incorporate various kinds of user-specific constraints
- Representative algorithms
  - GSP (Generalized Sequential Patterns): Srikant & Agrawal @ EDBT'96)
  - Vertical format-based mining: SPADE (Zaki@Machine Leanining'00)
  - Pattern-growth methods: PrefixSpan (Pei, Han et al. @ICDE'01)
- Constraint-based sequential pattern mining (SPIRIT: Garofalakis, Rastogi, Shim@VLDB'99; Pei, Han, Wang @ CIKM'02)
- Mining closed sequential patterns: CloSpan (Yan, Han et al. @SDM'03)

The Apriori Property of Sequential Patterns

- A basic property: Apriori (Agrawal & Sirkant'94)
  - If a sequence S is not frequent
  - Then none of the super-sequences of S is frequent
  - E.g, <hb> is infrequent → so do <hab> and <(ah)b>

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)>

Given *support threshold*
min_sup =2

GSP—Generalized Sequential Pattern Mining

- GSP (Generalized Sequential Pattern) mining algorithm
  - proposed by Agrawal and Srikant, EDBT'96
- 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

- Examine GSP using an example
- Initial candidates: all singleton sequences
  - <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

GSP: 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

The GSP Mining Process

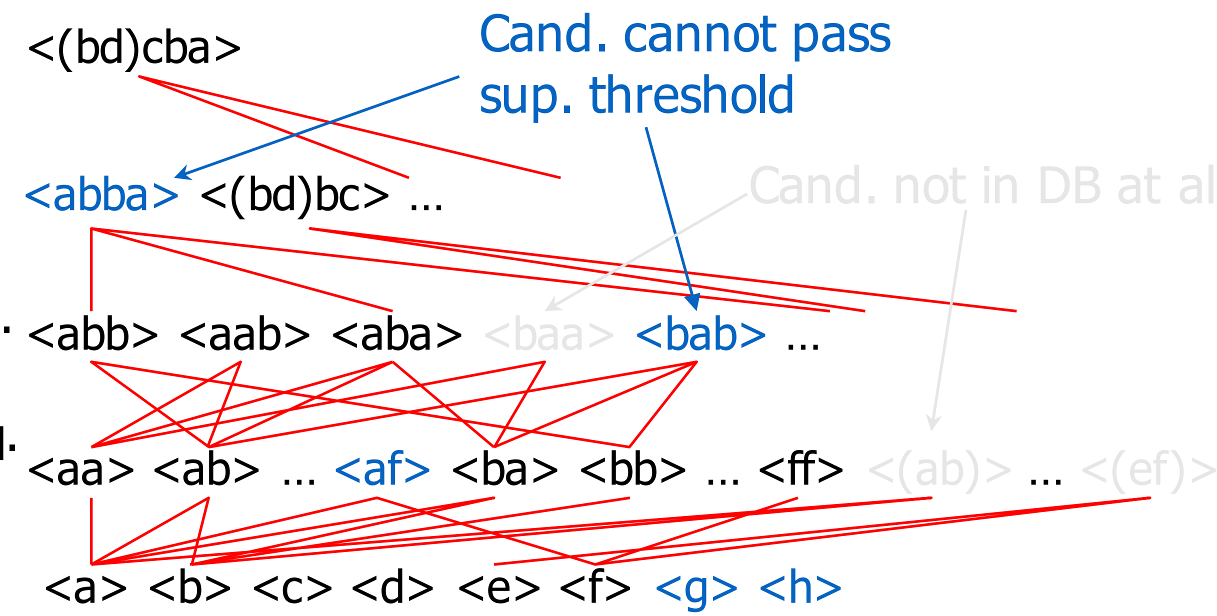
5th scan: 1 cand. 1 length-5 seq. pat.

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.



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)>

Bottlenecks of GSP

- A huge set of candidates could be generated
  - 1,000 frequent length-1 sequences generate s huge number of length-2 candidates!

$$\underbrace{1000 \times 1000}_{\text{Two itemsets}} + \underbrace{\frac{1000 \times 999}{2}}_{\text{One itemset}} = 1,499,500$$

- Multiple scans of database in mining
- Breadth-first search
- Mining long sequential patterns by growing from shorter patterns
  - Needs an exponential number of short candidates
  - A length-100 sequential pattern needs 10^{30} candidate sequences!

$$\sum_{i=1}^{100} \binom{100}{i} = 2^{100} - 1 \approx 10^{30}$$

PrefixSpan: Mining Sequential Patterns by Prefix Projections

Prefix	*Suffix* (Prefix-Based *Projection*)
<a>	<(abc)(ac)d(cf)>
<aa>	<(_bc)(ac)d(cf)>
<ab>	<(_c)(ac)d(cf)>

- Prefix and suffix
  - Given sequence <a(abc)(ac)d(cf)>
  - *Prefixes*: <a>, <aa>, <a(ab)> and <a(abc)>

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

- PrefixSpan Mining framework
  - Step 1: find length-1 sequential patterns
    - <a>, <b>, <c>, <d>, <e>, <f>
  - Step 2: divide search space and database. The complete set of seq. pat. can be partitioned into 6 subsets:
    - The ones having prefix <a>;
    - The ones having prefix <b>;
    - …
    - The ones having prefix <f>

Finding Seq. Patterns with Prefix <a>

- Only need to consider projections w.r.t. <a>
  - <a>-projected database:
    - <(abc)(ac)d(cf)>
    - <(_d)c(bc)(ae)>
    - <(_b)(df)cb>
    - <(_f)cbc>
- Find all the length-2 seq. pat. Having prefix <a>: <aa>, <ab>, <(ab)>, <ac>, <ad>, <af>
  - Further partition into 6 subsets
    - Having prefix <aa>;
    - …
    - Having prefix <af>

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

Completeness of PrefixSpan

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>

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

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

<u><b>-projected database</u>

…

… …

Having prefix <aa>

Having prefix <af>

<u><aa>-proj. db</u> … <u><af>-proj. db</u>

Major strength of PrefixSpan:

- No candidate sequence needs to be generated
- Projected (parititioned) databases keep shrinking

Speed-up by Pseudo-Projection

- Major cost of PrefixSpan: Constructing projected databases
  - Postfixes of sequences often appear repeatedly in recursive projected databases
- When (projected) database can be held in main memory, use pointers to form pseudo-projections
  - Pointer to the sequence
  - Offset of the postfix

s=<a(abc)(ac)d(cf)>

<a>

s|<a>: (, 2) <(abc)(ac)d(cf)>

<ab>

s|<ab>: (, 4) <(_c)(ac)d(cf)>

Pseudo-Projection vs. Physical Projection

- Pseudo-projection avoids physically copying postfixes
  - Efficient in running time and space when database can be held in main memory
- However, it is not efficient when database cannot fit in main memory
  - Disk-based random accessing is very costly
- Suggested Approach:
  - Integration of physical and pseudo-projection
  - Swapping to pseudo-projection when the data set fits in memory