Association Rule Mining

CS145

Fall 2015

Mining Various Kinds of Rules or Regularities

- Multi-level, quantitative association rules, correlation and causality, ratio rules, sequential patterns, emerging patterns, temporal associations, partial periodicity
- Classification, clustering, iceberg cubes, etc.

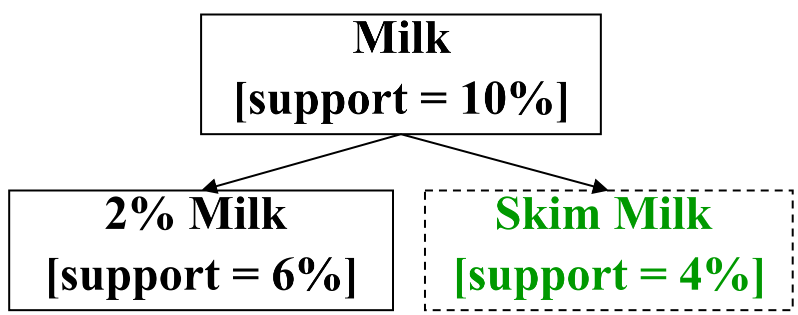
Multiple-level Association Rules

- Items often form hierarchy
- Flexible support settings: Items at the lower level are expected to have lower support.
- Transaction database can be encoded based on dimensions and levels
- explore shared multi-level mining

uniform support

Level 1
min_sup = 5%

Level 2
min_sup = 5%



reduced support

Level 1
min_sup = 5%

Level 2
min_sup = 3%

Multi-dimensional Association Rules

- Single-dimensional rules:
  - buys(X, "milk") ⇒ buys(X, "bread")
- MD rules: ≥ 2 dimensions or predicates
  - Inter-dimension assoc. rules (no repeated predicates)
    - age(X,"19-25") ∧ occupation(X,"student") ⇒ buys(X,"coke")
  - hybrid-dimension assoc. rules (repeated predicates)
    - age(X,"19-25") ∧ buys(X, "popcorn") ⇒ buys(X, "coke")
- Categorical Attributes: finite number of possible values, no order among values
- Quantitative Attributes: numeric, implicit order

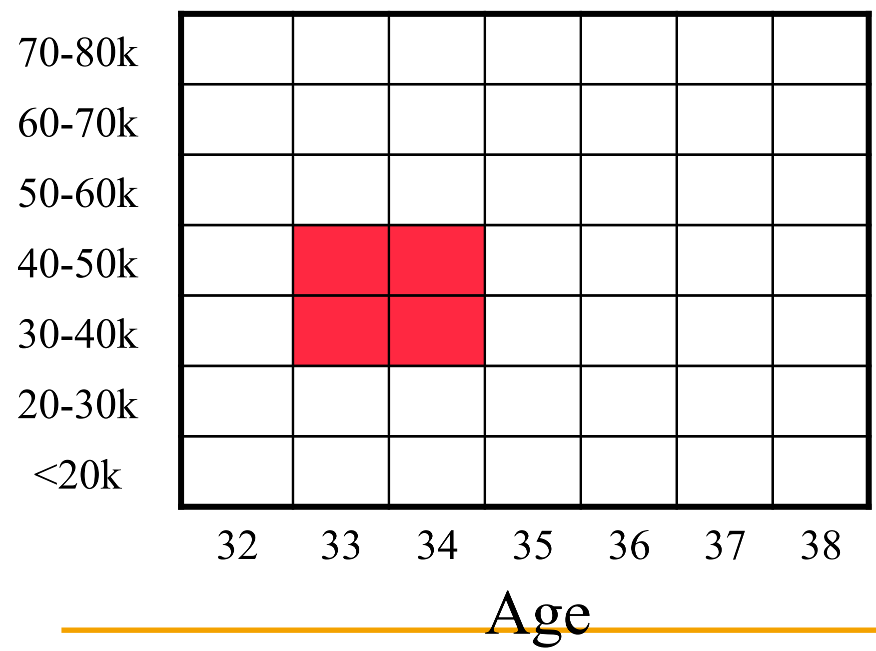
Quantitative/Weighted Association Rules

Numeric attributes are *dynamically* discretized

maximize the confidence or compactness of the rules

2-D quantitative association rules: $A_{quan1} \wedge A_{quan2} \Rightarrow A_{cat}$

Cluster "adjacent" association rules to form general rules using a 2-D grid.



age(X,"33-34") ∧ income(X,"30K - 50K") ⇒ buys(X,"high resolution TV")

Mining Distance-based Association Rules

- Binning methods do not capture semantics of interval data

Price	Equi-width	Equi-depth	Distance-based
7	[0,10]	[7,20]	[7,7]
20	[11,20]	[22,50]	[20,22]
22	[21,30]	[51,53]	
50	[31,40]		
51	[41,50]		[50,53]
53	[51,60		

- Distance-based partitioning
  - Density/number of points in an interval
  - "Closeness" of points in an interval

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

- 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
- <u>Rule (or pattern) constraint</u>
  - small sales (price < $10) triggers big sales (sum > $200)
- 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
- Data space pruning constraint
  - Data succinct: Data space can be pruned at the initial pattern mining process
  - Data anti-monotonic: If a transaction t does not satisfy c, t can be pruned from its further mining

Pattern Space Pruning with Anti-Monotonicity Constraints

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

- A constraint C is *anti-monotone* if the super pattern satisfies C, all of its sub-patterns do so too
- In other words, *anti-monotonicity:* If an itemset S **violates** the constraint, so does any of its superset
- Ex. 1. $sum(S.price) \le v$ is anti-monotone
- Ex. 2. range(S.profit) ≤ 15 is anti-monotone
  - Itemset *ab* violates C
  - So does every superset of *ab*
- Ex. 3. $sum(S.Price) \ge v$ is not anti-monotone
- Ex. 4. *support count* is anti-monotone: core property used in Apriori

Pattern Space Pruning with Monotonicity Constraints

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

- A constraint C is *monotone* if the pattern satisfies C, we do not need to check C in subsequent mining
- Alternatively, monotonicity: *If an itemset S **satisfies** the constraint, so does any of its superset*
- Ex. 1. $sum(S.Price) \geq v$ is monotone
- Ex. 2. $min(S.Price) \leq v$ is monotone
- Ex. 3. C: range(S.profit) ≥ 15
  - Itemset *ab* satisfies C
  - So does every superset of *ab*

Data Space Pruning with Data Anti-monotonicity

TDB (min_sup=2)

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

Item	Profit
a	40
b	0
c	-20
d	-15
e	-30
f	-10
g	20
h	-5

- A constraint c is *data anti-monotone* if, for a pattern p, it cannot be satisfied by a transaction t in p-projected database, it cannot be satisfied by t's projection on p's superset either
- The key for data anti-monotone is *recursive data reduction*
- Ex. 1. $sum(S.Price) \geq v$ is data anti-monotone
- Ex. 2. $min(S.Price) \leq v$ is data anti-monotone
- Ex. 3. C: $range(S.profit) \geq 25$ is data anti-monotone
  - Itemset {b}'s projected DB:
    - T10': {c, d, f, h}, T20': {c, d, f, g, h}, T30': {c, d, f, g}
  - C cannot be satisfied by T10', T10' can be pruned

Pattern Space Pruning with Succinctness

- Succinctness:
  - Given $A_{1,}$ the set of items satisfying a succinctness constraint *C*, then any set *S* satisfying *C* is based on A_1 , i.e., *S* contains a subset belonging to A_1
  - Idea: Without looking at the transaction database, whether an itemset *S* satisfies constraint C can be determined based on the selection of items
  - $min(S.Price) \le v$ is succinct
  - $sum(S.Price) \ge v$ 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:

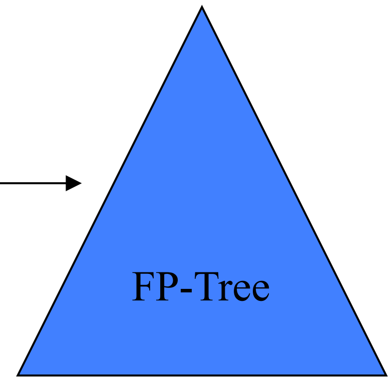
min{S.price } <= 1

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:

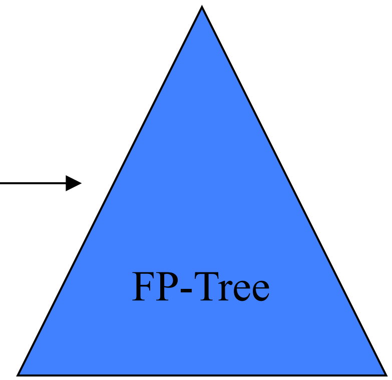
min{S.price } <= 1

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



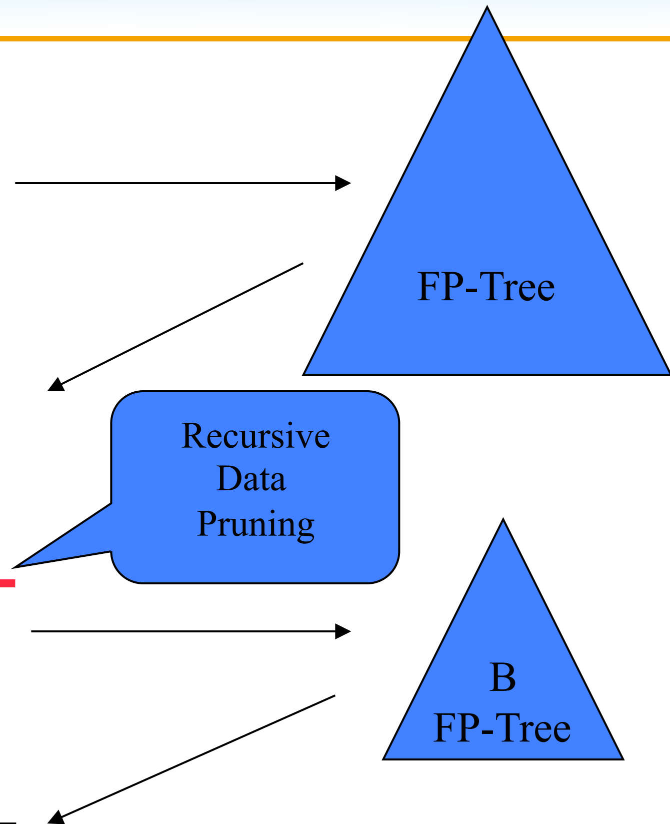
Single branch, we are done

Constraint:

min{S.price } <= 1

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

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



B-Projected DB

TID	Transaction
~~10~~	~~a, c, d, f, h~~
20	c, d, f, g, ~~h~~
30	c, d, f, g

Single branch:

bcdfg: 2

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

Item	Profit
a	40
b	0
c	-20
d	-15
e	-30
f	-10
g	20
h	-5

Constraint:
range{S.price } > 25
min_sup >= 2

Convertible Constraints: Ordering Data in Transactions

TDB (min_sup=2)

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

- Convert tough constraints 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!

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

Strongly Convertible Constraints

- avg(X) ≥ 25 is convertible anti-monotone w.r.t. 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 w.r.t. 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

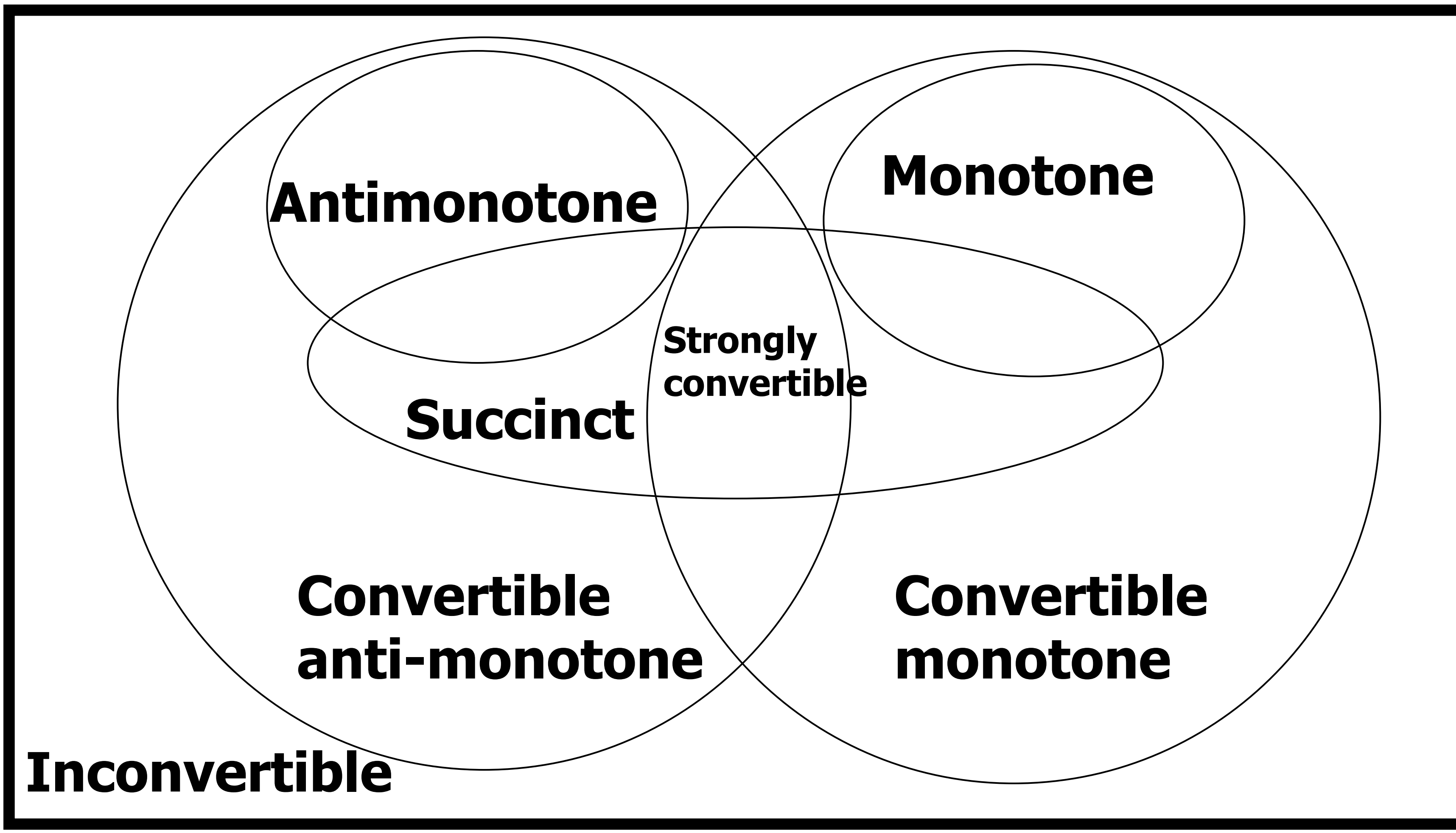
Constraint-Based Mining — A General Picture

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

What Constraints Are Convertible?

Constraint	Convertible anti-monotone	Convertible monotone	Strongly convertible
avg(S) ≤ , ≥ v	Yes	Yes	Yes
median(S) ≤ , ≥ v	Yes	Yes	Yes
sum(S) ≤ v (items could be of any value, v ≥ 0)	Yes	No	No
sum(S) ≤ v (items could be of any value, v ≤ 0)	No	Yes	No
sum(S) ≥ v (items could be of any value, v ≥ 0)	No	Yes	No
sum(S) ≥ v (items could be of any value, v ≤ 0)	Yes	No	No
……			

Classification of Constraints



Sequential Pattern Mining

- Why sequential pattern mining?
- GSP algorithm
- FreeSpan and PrefixSpan
- Boarder Collapsing
- Constraints and extensions

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*

Challenges on Sequential Pattern Mining

- A huge number of possible sequential patterns are hidden in databases
- A mining algorithm should
  - Find the complete set of patterns satisfying the minimum support (frequency) threshold
  - Be highly efficient, scalable, involving only a small number of database scans
  - Be able to incorporate various kinds of user-specific constraints

A Basic Property of Sequential Patterns: Apriori

- 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

Basic Algorithm : Breadth First Search (GSP)

- L=1
- While ($Result_L$!= NULL)
  - Candidate Generate
  - Prune
  - Test
  - L=L+1

Finding Length-1 Sequential Patterns

- 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

The Mining Process

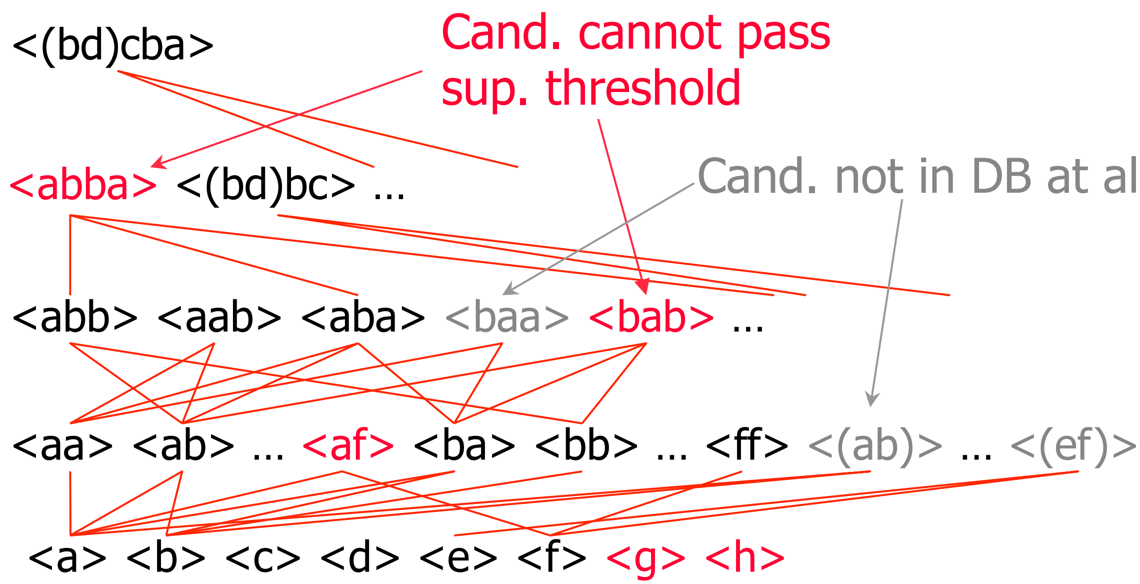
5th scan: 1 cand. 1 length-5 seq. pat.

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)>

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

Pattern Growth (prefixSpan)

- Prefix and Suffix (Projection)
  - <a>, <aa>, <a(ab)> and <a(abc)> are *prefixes* of sequence <a(abc)(ac)d(cf)>
  - Given sequence <a(abc)(ac)d(cf)>

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

Example

An Example

(min_sup=2):

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

Prefix	Sequential Patterns
<a>	<a>,<aa>,<ab><a(bc)>,<a(bc)a>,<aba>,<abc>,<(ab)>,<(ab)c>,<(ab)d>,<(ab)f>,<(ab)dc>,<ac>,<aca>,<acb>,<acc>,<ad>,<adc>,<af>
<b>	<b>, <ba>, <bc>, <(bc)>, <(bc)a>, <bd>, <bdc>,<bf>
<c>	<c>, <ca>, <cb>, <cc>
<d>	<d>,<db>,<dc>, <dcb>
<e>	<e>,<ea>,<eab>,<eac>,<eacb>,<eb>,<ebc>,<ec>,<ecb>,<ef>,<efb>,<efc>,<efcb>
<f>	<f>,<fb>,<fbc>, <fc>, <fcb>

PrefixSpan (the example to be continued)

Step1: Find length-1 sequential patterns;

<a>:4, <b>:4, <c>:4, <d>:3, <e>:3, <f>:3

support

pattern

Step2: Divide search space;

six subsets according to the six prefixes;

Step3: Find subsets of sequential patterns;

By constructing corresponding projected databases and mine each recursively.

Example to be continued

Sequence_id	Sequence	Projected(suffix) databases
10	<a(abc)(ac)d(cf)>	<a(abc)(ac)d(cf)>
20	<(ad)c(bc)(ae)>	<(ad)c(bc)(ae)>
30	<(ef)(ab)(df)cb>	<(ef)(ab)(df)cb>
40	<eg(af)cbc>	<eg(af)cbc>

Prefix	Projected(suffix) databases	Sequential Patterns
<a>	<(abc)(ac)d(cf)>, <(_d)c(bc)(ae)>, <(_b)(df)cb>, <(_f)cbc>	<a>,<aa>,<ab><a(bc)>,<a(bc)a>,<aba>,<abc>,<(ab)>,<(ab)c>,<(ab)d>,<(ab)f>,<(ab)dc>,<ac>,<aca>,<acb>,<acc>,<ad>,<adc>,<af>

Example

Find sequential patterns having prefix <a>:

1. Scan sequence database S once. Sequences in S containing <a> are projected w.r.t <a> to form the <a>-projected database.
2. Scan <a>-projected database once, get six length-2 sequential patterns having prefix <a> :

   <a>:2 , <b>:4, <(_b)>:2, <c>:4, <d>:2, <f>:2

   <aa>:2 , <ab>:4, <(ab)>:2, <ac>:4, <ad>:2, <af>:2
3. Recursively, all sequential patterns having prefix <a> can be further partitioned into 6 subsets. Construct respective projected databases and mine each.

   e.g. <aa>-projected database has two sequences :

   <(_bc)(ac)d(cf)> and <(_e)>.

PrefixSpan Algorithm

Main Idea: Use frequent prefixes to divide the search space and to project sequence databases. only search the relevant sequences.

PrefixSpan(α, i, S|α)

1. Scan S|α once, find the set of frequent items b such that
    - b can be assembled to the last element of α to form a sequential pattern; or
    - <b> can be appended to α to form a sequential pattern.
2. For each frequent item b, appended it to α to form a sequential pattern α', and output α';
3. For each α', construct α'-projected database S|α', and call PrefixSpan(α', i+1,S|α').