Basic Concepts: Frequent Itemsets (Patterns)

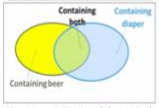
- Itemset: A set of one or more items
- k-itemset: X = {x1, …, xk}
- (absolute) support (count) of X: Frequency or the number of occurrences of an itemset X
- (relative) support, s: The fraction of transactions that contains X (i.e., the probability that a transaction contains X)
- An itemset X is *frequent* if the support of X is no less than a *minsup* threshold (denoted as σ)

Tid	Items bought
10	Beer, Nuts, Diaper
20	Beer, Coffee, Diaper
30	Beer, Diaper, Eggs
40	Nuts, Eggs, Milk
50	Nuts, Coffee, Diaper, Eggs, Milk

- Let *minsup* = 50%
- Freq. 1-itemsets:
  - Beer: 3 (60%); Nuts: 3 (60%)
  - Diaper: 4 (80%); Eggs: 3 (60%)
- Freq. 2-itemsets:
  - {Beer, Diaper}: 3 (60%)

From Frequent Itemsets to Association Rules

Tid	Items bought
10	Beer, Nuts, Diaper
20	Beer, Coffee, Diaper
30	Beer, Diaper, Eggs
40	Nuts, Eggs, Milk
50	Nuts, Coffee, Diaper, Eggs, Milk

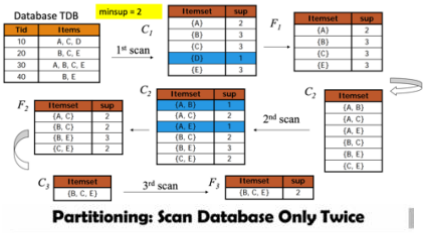


Note: Itemset: X ∪ Y, a subtle notation!

(Q: Are these all rules?)

- Association rules: X → Y (s, c)
  - Support, s: The probability that a transaction contains X ∪ Y
  - Confidence, c: The conditional probability that a transaction containing X also contains Y
  - c = sup(X ∪ Y) / sup(X)
- Association rule mining: Find all of the rules, X → Y, with minimum support and confidence
- Frequent itemsets: Let *minsup* = 50%
  - Freq. 1-itemsets: Beer: 3, Nuts: 3, Diaper: 4, Eggs: 3
  - Freq. 2-itemsets: {Beer, Diaper}: 3
- Association rules: Let *minconf* = 50%
  - Beer → Diaper (60%, 100%)
  - Diaper → Beer (60%, 75%)

The Apriori Algorithm—An Example



Partitioning: Scan Database Only Twice

- Theorem: *Any itemset that is potentially frequent in TDB must be frequent in at least one of the partitions of TDB*

- Method: (A. Savasere, E. Omiecinski and S. Navathe, *VLDB'95*)
  - Scan 1: Partition database (how?) and find local frequent patterns
  - Scan 2: Consolidate global frequent patterns (how to?)
- Why does this method guarantee to scan TDB only twice?

Mining Frequent Patterns, Associations, and Correlations

support(X ⇒ Y) = P(X UY),

confidence(X ⇒ Y) = P(Y|X).

$$confidence(A \Rightarrow B) = P(B|A) = \frac{support(A \cup B)}{support(A)} = \frac{support\_count(A \cup B)}{support\_count(A)}$$

-> An itemset X is closed in a data set D if there exists no proper super-itemset Y such that Y has the same support count as X in D.

-> An itemset X is a maximal frequent itemset (or max-itemset) in a data set D if X is frequent and there exists no frequent super pattern Y of X.

Exploring Vertical Data Format: ECLAT

- ECLAT (Equivalence Class Transformation): A depth-first search algorithm using set intersection [Zaki et al. @KDD'97]
- Tid-List: List of transaction-ids containing an itemset
- Vertical format: t(e) = {T10, T20, T30}; t(a) = {T10, T20}; t(ae) = {T10, T20}
- Properties of Tid-Lists
  - t(X) = t(Y): X and Y always happen together (e.g., t(ac) = t(d))
  - t(X) ⊂ t(Y): transaction having X always has Y (e.g., t(ac) ⊂ t(ce))
- Deriving frequent patterns based on vertical intersections
- Using diffset to accelerate mining
  - Only keep track of differences of tids
  - t(e) = {T10, T20, T30}, t(ce) = {T10, T30} → Diffset (ce, e) = {T20}

A transaction DB in Horizontal Data Format

Tid	Itemset
10	a, c, d, e
20	a, b, e
30	b, c, e

The transaction DB in Vertical Data Format

Item	TidList
a	10, 20
b	20, 30
c	10, 30
d	10
e	10, 20, 30

FPGrowth: Mining Frequent Patterns by Pattern Growth

- Idea: Frequent pattern growth (FPGrowth)
  - Find frequent single items and partition the database based on each such item
  - Recursively grow frequent patterns by doing the above for each partitioned database (also called *conditional database*)
  - To facilitate efficient processing, an efficient data structure, FP-tree, can be constructed
- Mining becomes
  - Recursively construct and mine (conditional) FP-trees
  - Until the resulting FP-tree is empty, or until it contains only one path—single path will generate all the combinations of its sub-paths, each of which is a frequent pattern

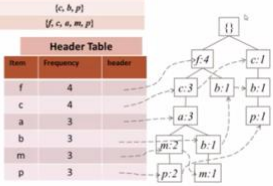
Example: Construct FP-tree from a Transational DB

TID	Items in the Transaction	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}

1. Scan DB once, find single item frequent pattern: Let min_support = 3
   f:4, a:3, c:4, b:3, m:3, p:3
2. Sort frequent items in frequency descending order, f-list
   F-list = f-c-a-b-m-p
3. Scan DB again, construct FP-tree

Header Table

Item	Frequency	header
f	4	
c	4	
a	3	
b	3	
m	3	
p	3	

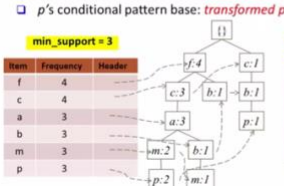


Divide and Conquer Based on Patterns and Data

- Pattern mining can be partitioned according to current patterns
  - Patterns containing p: p's conditional database: *fcam:2, cb:1*
  - Patterns having m but no p: m's conditional database: *fca:2, fcab:1*
  - ....
- *p*'s conditional pattern base: *transformed prefix paths* of item *p*

min_support = 3

Item	Frequency	Header
f	4	
c	4	
a	3	
b	3	
m	3	
p	3	



Conditional pattern bases

Item	Conditional 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

CLOSET+: Mining Closed Itemsets by Pattern-Growth

- Efficient, *direct* mining of closed itemsets
- Ex. Itemset merging: If Y appears in every occurrence of X, then Y is merged with X
  - d-proj. db: {acef, acf} → acfd-proj. db: {e}, thus we get: acfd:2
- Many other tricks (but not detailed here), such as
  - Hybrid tree projection
    - Bottom-up physical tree-projection

TID	Items
1	acdef
2	abe
3	cefg
4	acdf

Let minsupport = 2

a:3, c:3, d:2, e:3, f:3

F-List: a-c-e-f-d

Interestingness Measure: Lift

- Measure of dependent/correlated events: **lift**

$$lift(B,C) = \frac{c(B \to C)}{s(C)} = \frac{s(B \cup C)}{s(B) \times s(C)}$$

Lift is more telling than s & c

	B	¬B	Σrow
C	400	350	750
¬C	200	50	250
Σcol	600	400	1000

- Lift(B, C) may tell how B and C are correlated
  - Lift(B, C) = 1: B and C are independent
  - > 1: positively correlated
  - < 1: negatively correlated

Interestingness Measure: χ^2

- Another measure to test correlated events: χ^2

$$\chi^2 = \sum \frac{(Observed - Expected)^2}{Expected}$$

χ^2 tells also better than s & c

	B	¬B	Σrow
C	400 (450)	350 (300)	750
¬C	200 (150)	50 (100)	250
Σcol	600	400	1000

Expected value

Observed value

- General rules
  - χ^2 = 0: independent
  - χ^2 > 0: correlated, either positive or negative, so it needs additional test

Measure	Definition	Range	Null-Invariant
$\chi^2(A,B)$	$\sum_{i,j=0,1} \frac{(e(a_i b_j) - o(a_i b_j))^2}{e(a_i b_j)}$	$[0, \infty]$	No
$Lift(A,B)$	$\frac{s(A \cup B)}{s(A) \times s(B)}$	$[0, \infty]$	No
$AllConf(A,B)$	$\frac{s(A \cup B)}{max\{s(A), s(B)\}}$	$[0, 1]$	Yes
$Jaccard(A,B)$	$\frac{s(A \cup B)}{s(A)+s(B)-s(A \cup B)}$	$[0, 1]$	Yes
$Cosine(A,B)$	$\frac{s(A \cup B)}{\sqrt{s(A) \times s(B)}}$	$[0, 1]$	Yes
$Kulczynski(A,B)$	$\frac{1}{2}(\frac{s(A \cup B)}{s(A)} + \frac{s(A \cup B)}{s(B)})$	$[0, 1]$	Yes
$MaxConf(A,B)$	$max\{\frac{s(A)}{s(A \cup B)}, \frac{s(B)}{s(A \cup B)}\}$	$[0, 1]$	Yes

Many Null Transactions throws of Lift and Chi-square.

Imbalance Ratio with Kulczynski Measure

- IR (Imbalance Ratio): measure the imbalance of two itemsets A and B in rule implications:

$$IR(A,B) = \frac{|s(A) - s(B)|}{s(A) + s(B) - s(A \cup B)}$$

Data set	mc	¬mc	m ¬c	¬m ¬c	Jaccard	Cosine	Kulc	IR
D_1	10,000	1,000	1,000	100,000	0.83	0.91	0.91	0
D_2	10,000	1,000	1,000	100	0.83	0.91	0.91	0
D_3	100	1,000	1,000	100,000	0.05	0.09	0.09	0
D_4	1,000	1,000	1,000	100,000	0.33	0.5	0.5	0
D_5	1,000	100	10,000	100,000	0.09	0.29	0.5	0.89
D_6	1,000	10	100,000	100,000	0.01	0.10	0.5	0.99

Multi-Level Freq Patterns - Efficient mining: Shared multi-level mining - Use the lowest min-support to pass down the set of candidates

Compressed Patterns

Pat-ID	Item-Sets	Support
P1	{A, C, E, S}	205227
P2	{F, A, C, E, S}	205211
P3	{F, A, C, E, T, S}	101758
P4	{F, A, C, T, S}	161563
P5	{A, C, T, S}	161576

$$P_3 = \{F, A, C, E, T, S\}$$

$$Dist(P_4, P_3) = 1 - \frac{|T(P_4)|}{|T(P_3)|} = 1 - \frac{101758}{|T(P_3)|} < \delta = 0.4$$

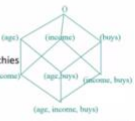
$$\frac{101758}{|T(P_3)|} > 0.6$$

$$Dist(P_1, P_3) = 1 - \frac{|T(P_1) \cap T(P_3)|}{|T(P_1) \cup T(P_3)|} = 1 - \frac{|T(P_3)|}{|T(P_1)|}$$

$$|T(P_3)| < \frac{101758}{0.6} = 169596.7$$

Mining Quantitative Associations

- Mining associations with numerical attributes
  - Ex.: Numerical attributes: age and salary
- Methods
  - Static discretization based on predefined concept hierarchies
    - Data cube-based aggregation
  - Dynamic discretization based on data distribution
  - Clustering: Distance-based association
    - First one-dimensional clustering, then association
  - Deviation analysis:
    - Gender = female ⇒ Wage: mean=$7/hr (overall mean = $9)

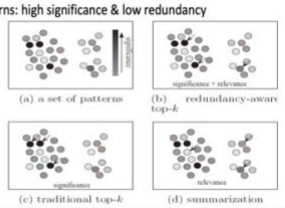


Mining Extraordinary Phenomena in Quantitative Association Mining

- Mining extraordinary (i.e., interesting) phenomena
  - Ex.: Gender = female ⇒ Wage: mean=$7/hr (overall mean = $9)
  - LHS: a subset of the population
  - RHS: an extraordinary behavior of this subset
- The rule is accepted only if a statistical test (e.g., Z-test) confirms the inference with high confidence
- Subrule: Highlights the extraordinary behavior of a subset of the population of the super rule
  - Ex.: (Gender = female) ^ (South ≠ yes) ⇒ mean wage = $6.3/hr
- Rule condition can be categorical or numerical (quantitative rules)
  - Ex.: Education in [14-18] (yrs) ⇒ mean wage = $11.64/hr
- Efficient methods have been developed for mining such extraordinary rules (e.g., Aumann and Lindell@KDD'99)

Redundancy-Aware Top-k Patterns

- Desired patterns: high significance & low redundancy



(a) a set of patterns (b) redundancy-aware top-k (c) traditional top-k (d) summarization

- Method: Use MMS (Maximal Marginal Significance) for measuring the combined significance of a pattern set
- Xin et al., Extracting Redundancy-Aware Top-K Patterns, KDD'06

Defining Negative Correlation: Need Null-Invariance in Definition

- A good definition on negative correlation should take care of the null-invariance problem
  - Whether two itemsets A and B are negatively correlated should not be influenced by the number of null-transactions
- A Kulczynski measure-based definition
  - If itemsets A and B are frequent but (P(A|B) + P(B|A))/2 < ε, where ε is a negative pattern threshold, then A and B are negatively correlated
- For the same needle package problem:
  - No matter there are in total 200 or 10^5 transactions
  - If ε = 0.01, we have (P(A|B) + P(B|A))/2 = (0.01 + 0.01)/2 < ε

Mining Compressed Patterns

Pat-ID	Item-Sets	Support
P1	{38,16,18,12}	205227
P2	{38,16,18,12,17}	205211
P3	{39,38,16,18,12,17}	101758
P4	{39,16,18,12,17}	161563
P5	{39,16,18,12}	161576

- Closed patterns
  - P1, P2, P3, P4, P5
  - Emphasizes too much on support
  - There is no compression
- Max-patterns
  - P3: information loss
- Desired output (a good balance):
  - P2, P3, P4

- Why mining compressed patterns?
  - Too many scattered patterns but not so meaningful
- Pattern distance measure

$$Dist(P_1, P_2) = 1 - \frac{|T(P_1) \cap T(P_2)|}{|T(P_1) \cup T(P_2)|}$$

- δ-clustering: For each pattern P, find all patterns which can be expressed by P and whose distance to P is within δ (δ-cover)
- All patterns in the cluster can be represented by P
- Method for efficient, direct mining of compressed frequent patterns (e.g., D. Xin, J. Han, X. Yan, H. Cheng, "On Compressing Frequent Patterns", Knowledge and Data Engineering, 60:5-29, 2007)

Sequential Pattern and Sequential Pattern Mining

- Sequential pattern mining: Given a set of sequences, find the complete set of *frequent* subsequences (i.e., satisfying the min_sup threshold)

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)cb>

- An element may contain a set of *items* (also called *events*)
- 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 Algorithms

- Algorithm requirement: Efficient, scalable, finding complete set, incorporating various kinds of user-specific constraints
- The Apriori property still holds: If a subsequence s_1 is infrequent, none of s_1's super-sequences can be frequent
- Representative algorithms
  - GSP (Generalized Sequential Patterns): Srikant & Agrawal @ EDBT'96)
  - Vertical format-based mining: SPADE (Zaki@Machine Leanining'00)
  - Pattern-growth methods: PrefixSpan (Pei, et al. @TKDE'04)
- Mining closed sequential patterns: CloSpan (Yan, et al. @SDM'03)
- Constraint-based sequential pattern mining

GSP: Apriori-Based Sequential Pattern Mining

- Initial candidates: All singleton sequences
  - <a>, <b>, <c>, <d>, <e>, <f>, <g>, <h>
- Scan DB once, count support for each candidate
- Generate length-2 candidate sequences

min_sup = 2

Cand.	sup
<a>	3
<b>	5
<c>	4
<d>	3
<e>	3
<f>	2
<g>	1
<h>	1

	<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>

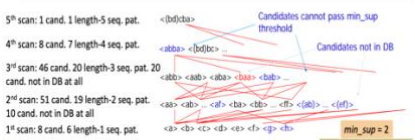
SID	Sequence
10	<(bd)cb(ac)>
20	<(bf)(ce)b(fg)>
30	<(ah)(bf)abf>
40	<(be)(ce)d>
50	<a(bd)bcb(ade)>

	<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>						

- Length-2 candidates: 36 + 15= 51
- Without Apriori pruning: 8*8+8*7/2=92 candidates

GSP (Generalized Sequential Patterns): Srikant & Agrawal @ EDBT'96)

GSP Mining and Pruning



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

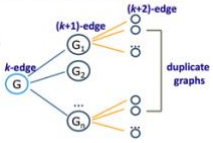
- Repeat (for each level (i.e., length-k))
  - Scan DB to find length-k frequent sequences
  - Generate length-(k+1) candidate sequences from length-k frequent sequences using Apriori
  - set k = k+1
- Until no frequent sequence or no candidate can be found

Implementation Consideration: Pseudo-Projection vs. Physical Projection

- Major cost of PrefixSpan: Constructing projected DBs
  - Suffixes largely repeating in recursive projected DBs
- When DB can be held in main memory, use pseudo projection
  - No physically copying suffixes
  - Pointer to the sequence
  - Offset of the suffix
- But if it does not fit in memory
  - Physical projection
- Suggested approach:
  - Integration of physical and pseudo-projection
  - Swapping to pseudo-projection when the data fits in memory

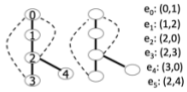
Pattern-Growth Approach

- Depth-first growth of subgraphs from k-edge to (k+1)-edge, then (k+2)-edge subgraphs
- Major challenge
  - Generating many duplicate subgraphs
- Major idea to solve the problem
  - Define an order to generate subgraphs
  - DFS spanning tree: Flatten a graph into a sequence using depth-first search
  - gSpan (Yan & Han, ICDM'02)



gSPAN: Graph Pattern Growth in Order

- **Right-most path extension** in subgraph pattern growth
  - Right-most path: The path from root to the right-most leaf (choose the vertex with the smallest index at each step)
  - Reduce generation of duplicate subgraphs
- **Completeness:** The enumeration of graphs using right-most path extension is complete
- DFS code: Flatten a graph into a sequence using depth-first search



Constraints in General Data Mining

A data mining query can be in the form of a meta-rule or with the following language primitives

- Knowledge type constraint
  - Ex.: Classification, association, clustering, outlier finding, ...
- Data constraint – using SQL-like queries
  - Ex.: Find products sold together in NY stores this year
- Dimension/level constraint
  - Ex.: In relevance to region, price, brand, customer category
- Rule (or pattern) constraint
  - Ex.: Small sales (price < $10) triggers big sales (sum > $200)
- Interestingness constraint
  - Ex.: Strong rules: min_sup ≥ 0.02, min_conf ≥ 0.6, min_correlation ≥ 0.7

Pattern Monotonicity and Its Roles

- A constraint c is *monotone*: If an itemset S **satisfies** the constraint c, so does any of its superset
  - That is, we do not need to check c in subsequent mining
- Ex. 1: c_1: *sum*(*S.Price*) ≥ *v* is monotone
- Ex. 2: c_2: *min*(*S.Price*) ≤ *v* is monotone
- Ex. 3: c_3: range(S.profit) ≥ 15 is monotone
  - Itemset *ab* satisfies c_3
  - So does every superset of *ab*

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

min_sup = 2
price(item)>0

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

Sequential Pattern Mining in Vertical Data Format: The SPADE Algorithm

- A sequence database is mapped to: <SID, EID>
- Grow the subsequences (patterns) one item at a time by Apriori candidate generation

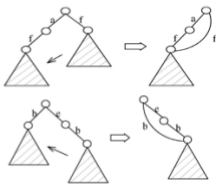
SID	Sequence
1	<a(abc)(ac)d(cf)>
2	<(ad)c(bc)(ae)>
3	<(ef)(ab)(df)cb>
4	<eg(af)cbc>

min_sup = 2

Ref: SPADE (Sequential PAttern Discovery using Equivalent Class) [M. Zaki 2001]

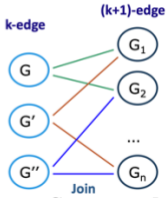
CloSpan: Mining Closed Sequential Patterns

- A closed sequential pattern *s*: There exists no superpattern *s'* such that *s'* ⊃ *s*, and *s'* and *s* have the same support
- Which ones are closed? <abc>: 20, <abcd>:20, <abcde>: 15
- Why directly mine closed sequential patterns?
  - Reduce # of (redundant) patterns
  - Attain the same expressive power
- Property P_1: If *s* ⊃ s_1, *s* is closed iff two project DBs have the same size
- Explore *Backward Subpattern* and *Backward Superpattern* pruning to prune redundant search space
- Greatly enhances efficiency (Yan, et al., SDM'03)



Apriori-Based Approach

- The Apriori property (anti-monotonicity): A size-k subgraph is frequent if and only if all of its subgraphs are frequent
- A candidate size-(k+1) edge/vertex subgraph is generated if its corresponding two k-edge/vertex subgraphs are frequent
- Iterative mining process:
  - Candidate-generation → candidate pruning → support counting → candidate elimination



SpiderMine: Mining Top-K Large Structural Patterns in a Massive Network

- Large patterns are informative to characterize a large network (e.g., social network, web, or bio-network)
- Similar to pattern fusion, mining large patterns should not aim for completeness but for representativeness of the target results
- SpiderMine (Zhu et al., VLDB'11): Mine top-K largest frequent substructure patterns whose diameter is bounded by D_{max} with a probability at least 1−ϵ
- General idea: Large patterns are composed of a number of small components ("spiders"), which will eventually connect together after some rounds of pattern growth
- **r-Spider:** An r-spider is a frequent graph pattern P such that there exists a vertex u of P, and all other vertices of P are within distance r from u

Pattern Space Pruning with Pattern Anti-Monotonicity

- Constraint c is *anti-monotone*
  - If an itemset S **violates** constraint c, so does any of its superset
  - That is, mining on itemset S can be terminated
- Ex. 1: c_1: *sum*(*S.price*) ≤ *v* is anti-monotone
- Ex. 2: c_2: range(S.profit) ≤ 15 is anti-monotone
  - Itemset *ab* violates c_2 (range(ab) = 40)
  - So does every superset of *ab*
- Ex. 3. c_3: *sum*(*S.Price*) ≥ *v* is not anti-monotone
- Ex. 4. Is c_4: *support*(S) ≥ σ anti-monotone?
  - Yes! Apriori pruning is essentially pruning with an anti-monotonic constraint!

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

min_sup = 2
price(item)>0

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

PrefixSpan: A Pattern-Growth Approach

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

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

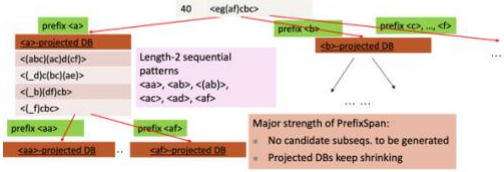
- Prefix and suffix
  - Given <a(abc)(ac)d(cf)>
  - Prefixes: <a>, <aa>, <a(ab)>, <a(abc)>, ...
  - Suffix: Prefixes-based projection
- PrefixSpan Mining: Prefix Projections
  - Step 1: Find length-1 sequential patterns
    - <a>, <b>, <c>, <d>, <e>, <f>
  - Step 2: Divide search space and mine each projected DB
    - <a>-projected DB,
    - <b>-projected DB,
    - ...
    - <f>-projected DB, ...

PrefixSpan (Prefix-projected Sequential pattern mining) Pei, et al. @TKDE'04

PrefixSpan: Mining Prefix-Projected DBs

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

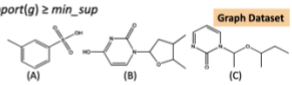
Length-1 sequential patterns
<a>, <b>, <c>, <d>, <e>, <f>



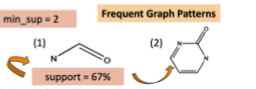
Major strength of PrefixSpan:
- No candidate subseqs. to be generated
- Projected DBs keep shrinking

Frequent (Sub)Graph Patterns

- Given a labeled graph dataset D = {G_1, G_2, ..., G_n}, the supporting graph set of a subgraph g is D_g = {G_i | $g \subseteq G_i$, $G_i \in D$}
  - support(g) = $|D_g|/|D|$
- A (sub)graph g is ***frequent*** if *support*(g) ≥ *min_sup*
- Ex.: Chemical structures
- Alternative:
  - Mining frequent subgraph patterns from a single large graph or network

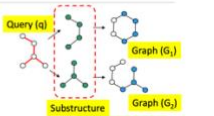


min_sup = 2



Feature-Based Similarity Search

- Decompose a query graph into a set of features
- Feature-based similarity measure
  - Each graph is represented as a feature vector X = {x_1, x_2, ..., x_n}
  - Similarity is defined by the distance of their corresponding vectors
- If graph G contains the major part of a query graph q, G should share a number of common features with q
  - Given a relaxation ratio, one can calculate the maximal number of features that can be missed!



Graphs in database

features	G1	G2	G3	G4	G5
f1	0	1	0	1	1
f2	0	1	0	0	1
f3	1	0	1	1	1
f4	1	0	0	0	1
f5	0	0	1	1	0

A feature-graph matrix

Assume: Query graph has 5 features
Relaxation threshold: Can miss at most 2 features
Then: G_3, G_4, G_5 are pruned

Different Kinds of Constraints Lead to Different Pruning Strategies

- Constraints can be categorized as
  - Pattern space pruning constraints vs. data space pruning constraints
- 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: If the constraint c can be enforced by directly manipulating the data
  - Convertible: c can be converted to monotonic or anti-monotonic if items can be properly ordered in processing
- Data space pruning constraints
  - Data succinct: Data space can be pruned at the initial pattern mining process
  - Data anti-monotonic: If a transaction t does not satisfy c, then t can be pruned to reduce data processing effort

Data Space Pruning with Data Anti-Monotonicity

- A constraint c is *data anti-monotone*: In the mining process, if a data entry t cannot satisfy a pattern p under c, t cannot satisfy p's superset either
  - Data space pruning: Data entry t can be pruned
- Ex. 1: c_1: *sum*(*S.Profit*) ≥ *v* is data anti-monotone
  - Let constraint c_1 be: *sum*(*S.Profit*) ≥ 25
  - T_{30}: {b, c, d, f, g} can be removed since none of their combinations can make an S whose sum of the profit is ≥ 25
- Ex. 2: c_2: *min*(*S.Price*) ≤ *v* is data anti-monotone
  - Consider *v* = 5 but every item in a transaction, say T_{50}, has a price higher than 10
- Ex. 3: c_3: *range*(*S.Profit*) > *25* is data anti-monotone

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

min_sup = 2
price(item) > 0

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