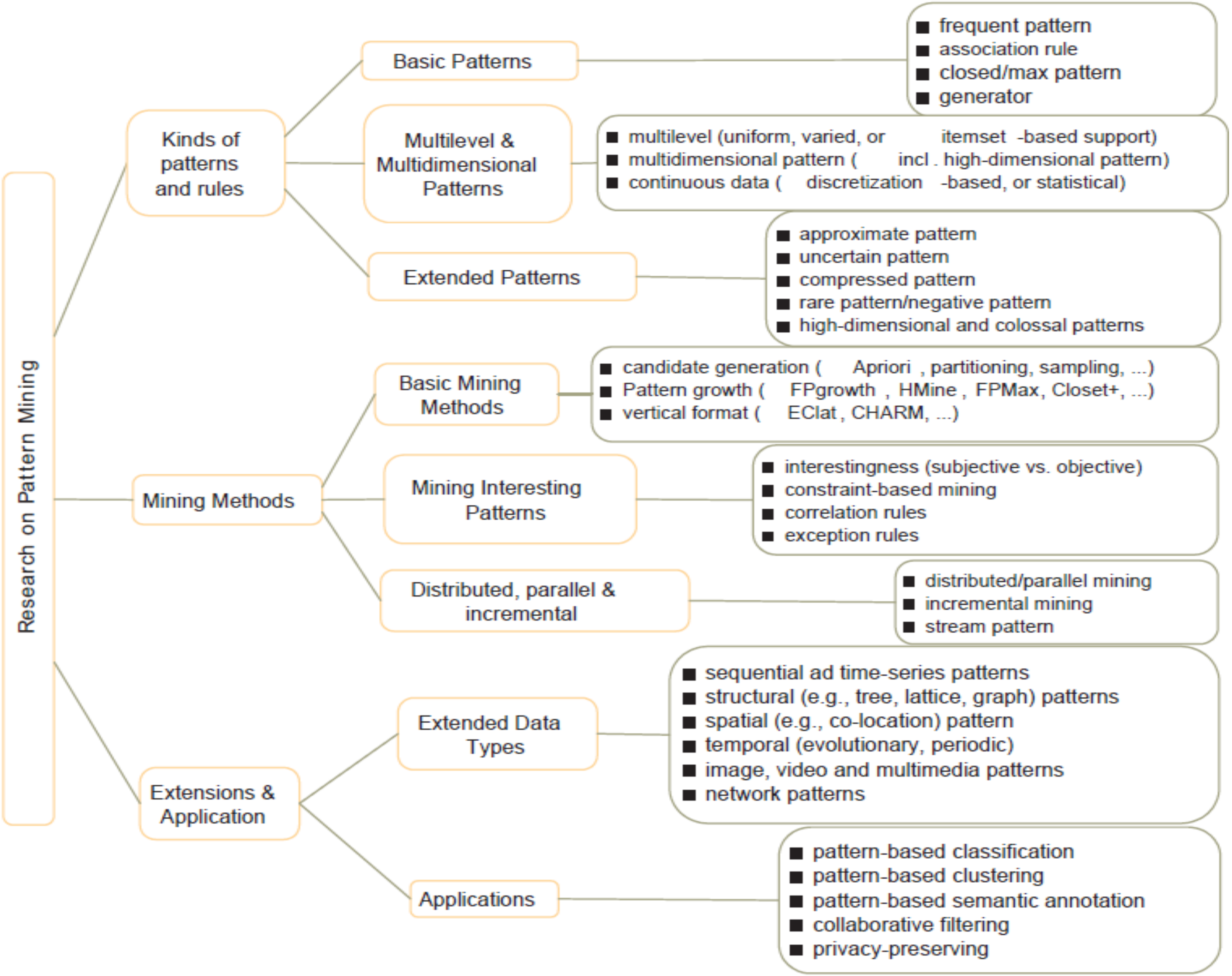
Chapter 7. Advanced Frequent Pattern Mining

Meng Jiang

CS412 Summer 2017:

Introduction to Data Mining

Research on Pattern Mining: A Road Map



Advanced Frequent Pattern Mining

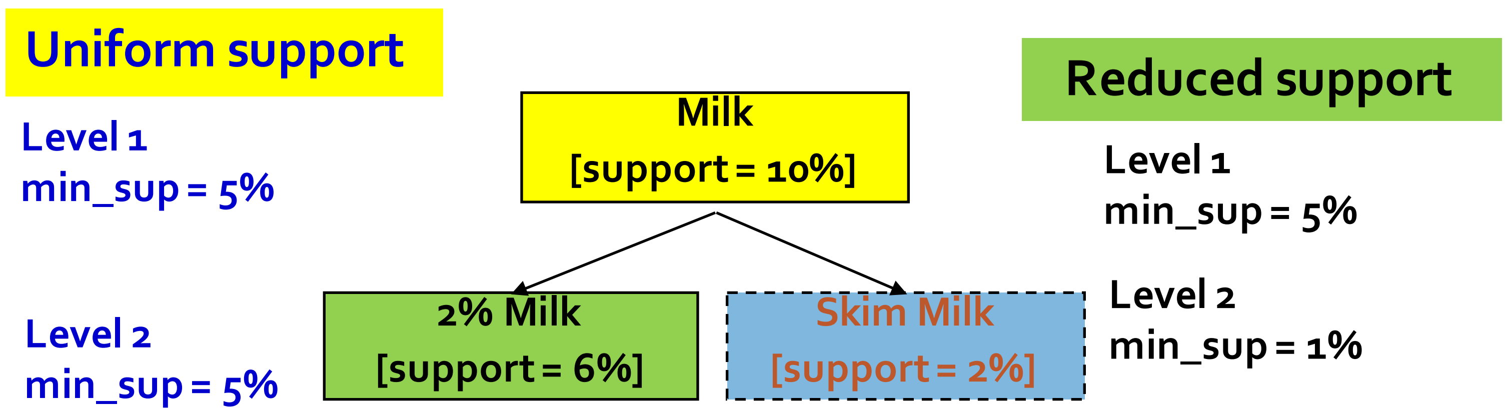
- **Mining Diverse Patterns**
- Constraint-Based Frequent Pattern Mining
- Mining High-Dimensional Data and Colossal Patterns
- Sequential Pattern Mining
- Graph Pattern Mining

Mining Diverse Patterns

- Mining Multiple-Level Associations
- Mining Multi-Dimensional Associations
- Mining Quantitative Associations
- Mining Negative Correlations
- Mining Compressed Patterns

Mining Multiple-Level Frequent Patterns

- Items often form hierarchies
  - Ex.: Dairyland 2% milk; Wonder wheat bread
- How to set min-support thresholds?
  - Uniform min-support across multiple levels (reasonable?)
  - Level-reduced min-support: Items at the lower level are expected to have lower support



Redundancy Filtering at Mining Multi-Level Associations

- Multi-level association mining may generate many redundant rules
- Redundancy filtering: Some rules may be redundant due to "ancestor" relationships between items

  (Suppose the 2% milk sold is about ¼ of milk sold in gallons)

  – milk $\Rightarrow$ wheat bread [support = 8%, confidence = 70%] (1)

  – 2% milk $\Rightarrow$ wheat bread [support = 2%, confidence = 72%] (2)

- A rule is *redundant* if its support is close to the "expected" value, according to its "ancestor" rule, and it has a similar confidence as its "ancestor"
  – Rule (1) is an ancestor of rule (2), which one to prune?

Customized Min-Supports for Different Kinds of Items

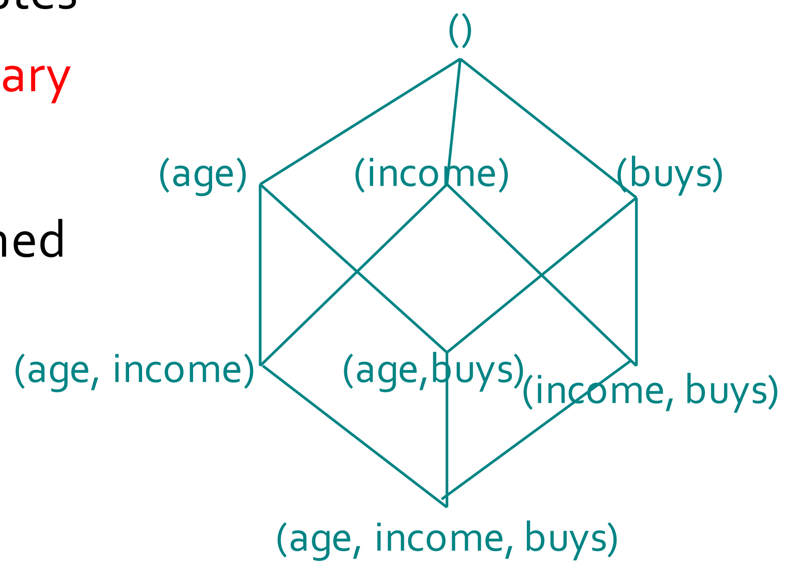
- We have used the same min-support threshold for all the items or item sets to be mined in each association mining
- In reality, some items (e.g., diamond, watch, …) are valuable but less frequent
- It is necessary to have customized min-support settings for different kinds of items
- One Method: Use group-based "individualized" min-support
  - E.g., {diamond, watch}: 0.05%; {bread, milk}: 5%; …

Mining Multi-Dimensional Associations

- Single-dimensional rules (e.g., items are all in "product" dimension)
  - buys(X, "milk") ⇒ buys(X, "bread")
- Multi-dimensional rules (i.e., items in ≥ 2 dimensions or predicates)
  - Inter-dimension association rules (*no repeated predicates*)
    - age(X, "18-25") ∧ occupation(X, "student") ⇒ buys(X, "coke")
  - Hybrid-dimension association rules (*repeated predicates*)
    - age(X, "18-25") ∧ buys(X, "popcorn") ⇒ buys(X, "coke")
- Attributes can be categorical or numerical
  - Categorical Attributes (e.g., *profession*, *product*: no ordering among values): Data cube for inter-dimension association
  - Quantitative Attributes: Numeric, implicit ordering among values—discretization, clustering, and gradient approaches

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)

Last Lecture

- Mining multi-level frequent patterns
  - Reduced support
  - Redundancy filtering
  - Group-based individualized min-support
- Mining multi-dimensional association rules
  - X→Y, variables. X: antecedent (condition) or left-hand-side (LHS); Y: consequent or right-hand-side (RHS)
  - Inter-dimension, hybrid-dimension
  - Attributes: categorical, numerical
- Mining quantitative association rules
  - Mean, deviation, minimum, maximum as variables: Data cube
- TODAY: Mining negative patterns, mining compressed patterns…

Rare Patterns vs. Negative Patterns

- Rare patterns
  - Very low support but interesting (e.g., buying Rolex watches)
  - How to mine them? Setting individualized, group-based min-support thresholds for different groups of items
- Negative patterns
  - Negatively correlated: Unlikely to happen together
  - Ex.: Since it is unlikely that the same customer buys both a Ford Expedition (an SUV car) and a Ford Fusion (a hybrid car), buying a Ford Expedition and buying a Ford Fusion are likely negatively correlated patterns
  - How to define negative patterns?

Defining Negative Correlated Patterns

- A support-based definition
  - If itemsets A and B are both frequent but rarely occur together, i.e., sup(A U B) << sup (A) × sup(B)
  - Then A and B are negatively correlated

Does this remind you the definition of *lift*?

- Is this a good definition for large transaction datasets?
- Ex.: Suppose a store sold two needle packages A and B 100 times each, but only one transaction contained both A and B
  - When there are in total 200 transactions, we have
    - s(A U B) = 0.005, s(A) × s(B) = 0.25, s(A U B) << s(A) × s(B)
  - But when there are 10^5 transactions, we have
    - s(A U B) = $1/10^5$, s(A) × s(B) = $1/10^3 \times 1/10^3$, s(A U B) > s(A) × s(B)
  - What is the problem? — Null transactions: The support-based definition is not null-invariant!

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 $(P(A|B) + P(B|A))/2 < \epsilon$, 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 $\epsilon = 0.01$, we have $(P(A|B) + P(B|A))/2 = (0.01 + 0.01)/2 < \epsilon$

Mining Compressed Patterns

- 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)

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

Advanced Frequent Pattern Mining

- Mining Diverse Patterns
- **Constraint-Based Frequent Pattern Mining**
- Mining High-Dimensional Data and Colossal Patterns
- Sequential Pattern Mining
- Graph Pattern Mining

Why Constraint-Based Mining?

- Finding all the patterns in a dataset autonomously? — unrealistic!
  - Too many patterns but not necessarily user-interested!
- Pattern 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

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: $min\_sup \geq 0.02$, $min\_conf \geq 0.6$, $min\_correlation \geq 0.7$

Meta-Rule Guided Mining

- A meta-rule can contain partially instantiated predicates & constants
  - $P_1(X, Y) \wedge P_2(X, W) \Rightarrow buys(X, \text{"iPad"})$
- The resulting mined rule can be
  - age(X, "15-25") ^ profession(X, "student") ⇒ buys(X, "iPad")
- In general, (meta) rules can be in the form of
  - $P_1 \wedge P_2 \wedge \ldots \wedge P_l \Rightarrow Q_1 \wedge Q_2 \wedge \ldots \wedge Q_r$
- Method to find meta-rules
  - Find frequent (l + r) predicates (based on *min-support*)
  - Push constants deeply when possible into the mining process
  - Also, push min_conf, min_correlation, and other measures as early as possible (measures acting as constraints)

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 (=no superset)
  - Monotonic: If c is satisfied, no need to check c again (=all supersets)
  - Succinct: if the constraint c can be enforced by directly manipulating the data (e.g., data pruning)
  - 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

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: c1: sum(S.price) ≤ v is anti-monotone
- Ex. 2: c2: range(S.profit) ≤ 15 is anti-monotone
  - Itemset ab violates c2 (range(ab) = 40)
  - So does every superset of ab
- Ex. 3. c3: sum(S.Price) ≥ v is not anti-monotone
- Ex. 4. Is c4: 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

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: c1: sum(S.Price) ≥ v is monotone
- Ex. 2: c2: min(S.Price) ≤ v is monotone
- Ex. 3: c3: range(S.profit) ≥ 15 is monotone
  - Itemset ab satisfies c3
  - 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

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 transaction 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

Succinctness: Pruning Both Data and Pattern Spaces

- Succinctness: if the constraint c can be enforced by directly manipulating the data
- Ex. 1: To find those patterns without item i
  - Remove i from DB and then mine (pattern space pruning)
- Ex. 2: To find those patterns containing item i
  - Mine only i-projected DB (data space pruning)
- Ex. 3: c_3: min(S.Price) ≤ v is succinct
  - Start with only items whose price ≤ v (pattern space pruning) and remove transactions with high-price items only (data space pruning)
- Ex. 4: c_4: sum(S.Price) ≥ v is not succinct
  - It cannot be determined beforehand since sum of the price of itemset S keeps increasing

Convertible Constraints: Ordering Data in Transactions

- Convert tough constraints into (anti-) monotone by proper ordering of items in transactions
- Examine c1: avg(S.profit) > 20
  - Order items in value-descending order
    - <a, g, f, h, b, d, c, e>
  - An itemset ab violates c1 (avg(ab) = 20)
    - So does ab* (i.e., ab-projected DB)
    - C1: anti-monotone if patterns grow in the right order!
- Can item-reordering work for Apriori?
  - Does not work for level-wise candidate generation!
  - avg(agf) = 23.3 > 20, but avg(gf) = 15 < 20

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

min_sup = 2

price(item)>0

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

How to Handle Multiple Constraints?

- It is beneficial to use multiple constraints in pattern mining
- But different constraints may require potentially conflicting item-ordering
  - If there exists an order R making both c1 and c2 convertible, try to sort items in the order that benefits pruning most
  - If there exists conflict ordering between c1 and c2
    - Try to sort data and enforce one constraint first (which one?)
    - Then enforce the other when mining the projected databases
- Ex. c_1: avg(S.profit) > 20, and c_2: avg(S.price) < 50
  - Sorted in profit descending order and use c1 first (assuming c_1 has more pruning power)
  - For each project DB, sort trans. in price ascending order and use c_2 at mining

Advanced Frequent Pattern Mining

- Mining Diverse Patterns
- Constraint-Based Frequent Pattern Mining
- **Mining High-Dimensional Data and Colossal Patterns**
- Sequential Pattern Mining
- Graph Pattern Mining

Mining Long Patterns: Challenges

- Mining long patterns is needed in bioinformatics, social network analysis, software engineering, …
  - But the methods introduced so far mine only short patterns (e.g., length < 10)
- Challenges of mining long patterns
  - The curse of "downward closure" property of frequent patterns
    - Any sub-pattern of a frequent pattern is frequent
    - If $\{a_1, a_2, \ldots, a_{100}\}$ is frequent, then $\{a_1\}, \{a_2\}, \ldots, \{a_{100}\}, \{a_1, a_2\}, \{a_1, a_3\}, \ldots, \{a_1, a_{100}\}, \{a_1, a_2, a_3\}, \ldots$ are all frequent! There are about 2^{100} such frequent itemsets!
  - No matter searching in breadth-first (e.g., Apriori) or depth-first (e.g., FPgrowth), if we still adopt the "small to large" step-by-step growing paradigm, we have to examine so many patterns, which leads to combinatorial explosion!

Colossal Patterns: A Motivating Example

- Let min-support σ= 20
- # of closed/maximal patterns of size 20: about $\binom{40}{20}$
- But there is only one pattern with size close to 40 (*i.e.*, *long* or *colossal*)
  - α= {41,42,…,79} of size 39
- Q: How to find it without generating an exponential number of size-20 patterns?

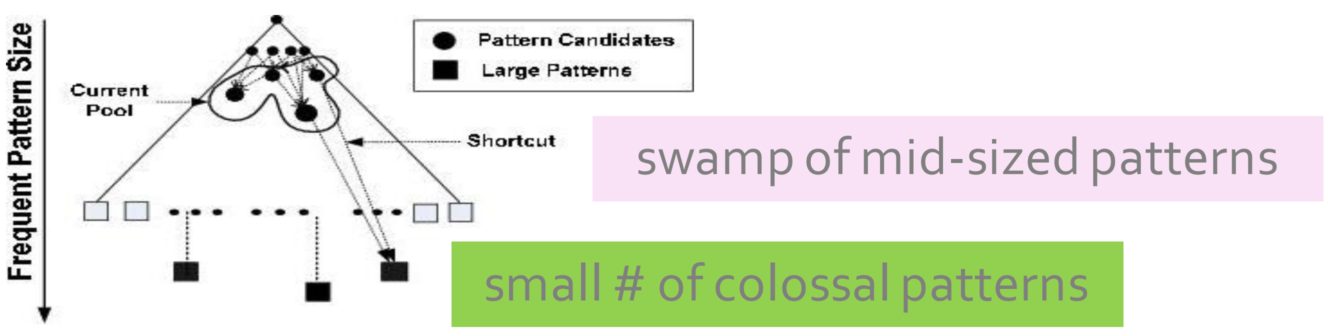
T_1 = 2 3 4 ….. 39 40
T_2 = 1 3 4 ….. 39 40
: .
: .
T_{40}=1 2 3 4 …… 39
T_{41}= 41 42 43 ….. 79
T_{42}= 41 42 43 ….. 79
: .
: .
T_{60}= 41 42 43 … 79

The existing fastest mining algorithms (*e.g.*, FPClose, LCM) fail to complete running

A new algorithm, *Pattern-Fusion*, outputs this colossal pattern in seconds

What Is Pattern-Fusion?

- Not strive for completeness (why?)
- Jump out of the swamp of the mid-sized intermediate "results"
- Strive for mining almost complete and representative colossal patterns: identify "short-cuts" and take "leaps"
- Key observation
  - The larger the pattern or the more distinct the pattern, the greater chance it will be generated from small ones
- Philosophy: Collection of small patterns hints at the larger patterns
- Pattern fusion strategy ("not crawl but jump"): Fuse small patterns together in one step to generate new pattern candidates of significant sizes



Observation: Colossal Patterns and Core Patterns

- Suppose dataset D contains 4 colossal patterns (below) plus many small patterns
  - $\{a_1, a_2, \ldots, a_{50}\}$: 40, $\{a_3, a_6, \ldots, a_{99}\}$: 60, $\{a_5, a_{10}, \ldots, a_{95}\}$: 80, $\{a_{10}, a_{20}, \ldots, a_{100}\}$: 100
- If you check the pattern pool of size-3, you may likely find
  - $\{a_2, a_4, a_{45}\}$: ~40; $\{a_3, a_{34}, a_{39}\}$: ~40; …, $\{a_5, a_{15}, a_{85}\}$: ~80, …, $\{a_{20}, a_{40}, a_{85}\}$: ~80, …
- If you merge the patterns with similar support, you may obtain candidates of much bigger size and easily validate whether they are true patterns
- *Core patterns* of a colossal pattern α: A set of subpatterns of α that cluster around α by sharing a similar support
- A colossal pattern has far more core patterns than a small-sized pattern
- A random draw from a complete set of pattern of size c would be more likely to pick a core pattern (or its descendant) of a colossal pattern
- A colossal pattern can be generated by merging a set of core patterns

Robustness of Colossal Patterns

- Core Patterns: For a frequent pattern α, a subpattern β is a τ-core pattern of α if β shares a similar support set with α, i.e.,

$$\frac{|D_\alpha|}{|D_\beta|} \geq \tau \qquad 0 < \tau \leq 1$$

where τ is called the core ratio

- (d, τ)-robustness: A pattern α is *(d, τ)-robust* if *d* is the maximum number of items that can be removed from α for the resulting pattern to remain a τ-core pattern of α
- For a (d, τ)-robust pattern α, it has $\Omega(2^d)$ core patterns
- Robustness of Colossal Patterns: A colossal pattern tends to have much more core patterns than small patterns
- Such core patterns can be clustered together to form "dense balls" based on pattern distance defined by

A random draw in the pattern space will hit somewhere in the ball with high probability

$$Dist(\alpha, \beta) = 1 - \frac{|D_\alpha \cap D_\beta|}{|D_\alpha \cup D_\beta|}$$

The Pattern-Fusion Algorithm

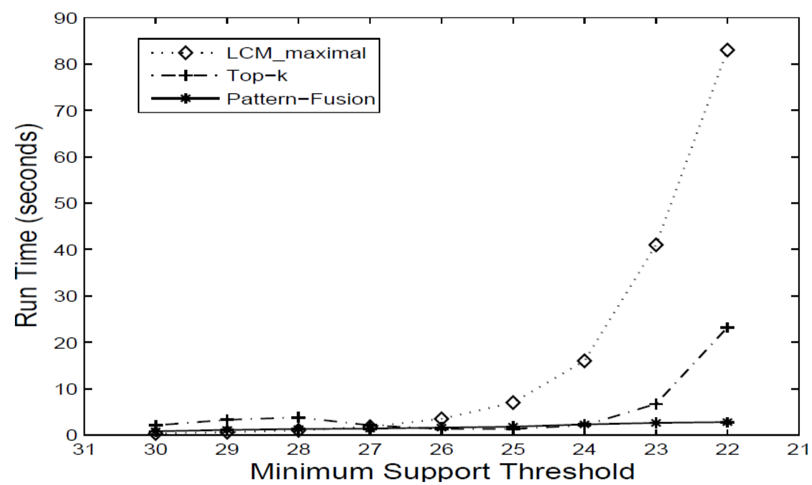
- Initialization (Creating initial pool): Use an existing algorithm to mine all frequent patterns up to a small size, e.g., 3
- Iteration (Iterative Pattern Fusion):
  - At each iteration, K seed patterns are randomly picked from the current pattern pool
  - For each seed pattern thus picked, we find all the patterns within a bounding ball centered at the seed pattern
  - All these patterns found are fused together to generate a set of super-patterns
  - All the super-patterns thus generated form a new pool for the next iteration
- Termination: when the current pool contains no more than K patterns at the beginning of an iteration

Experimental Results on Data Set: ALL

- ALL: A popular gene expression clinical data set on ALL-AML leukemia, with 38 transactions, each with 866 columns. There are 1,736 items in total.
  - When minimum support is high (e.g., 30), Pattern-Fusion gets all the largest colossal patterns with size greater than 85

Pattern Size	110	107	102	91	86	84	83
The complete set	1	1	1	1	1	2	6
Pattern-Fusion	1	1	1	1	1	1	4
Pattern Size	82	77	76	75	74	73	71
The complete set	1	2	1	1	1	2	1
Pattern-Fusion	0	2	0	1	1	1	1

Mining colossal patterns on a Leukemia dataset



Algorithm runtime comparison on another dataset

Advanced Frequent Pattern Mining

- Mining Diverse Patterns
- Constraint-Based Frequent Pattern Mining
- Mining High-Dimensional Data and Colossal Patterns
- **Sequential Pattern Mining**
- Graph Pattern Mining

Sequence Databases and Sequential Patterns

- Sequential pattern mining has broad applications
  - Customer shopping sequences
    - Purchase a laptop first, then a digital camera, and then a smartphone, within 6 months
  - Medical treatments, natural disasters (e.g., earthquakes), science & engineering processes, stocks and markets, …
  - Weblog click streams, calling patterns, …
  - Software engineering: Program execution sequences, …
  - Biological sequences: DNA, protein, …
- Transaction DB, sequence DB vs. time-series DB
- Gapped vs. non-gapped sequential patterns
  - Shopping sequences, clicking streams vs. biological sequences

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) c b >

- 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

http://hanj.cs.illinois.edu/pdf/span01.pdf

Sequence vs Element/Itemset/Event vs Item/Instance

Let $I = \{i_1, i_2, \ldots, i_n\}$ be a set of all **items**. An **itemset** is a subset of items. A **sequence** is an ordered list of itemsets. A sequence s is denoted by $\langle s_1 s_2 \cdots s_l \rangle$, where s_j is an itemset, i.e., $s_j \subseteq I$ for $1 \leq j \leq l$. s_j is also called an **element** of the sequence, and denoted as $(x_1 x_2 \cdots x_m)$, where x_k is an item, i.e., $x_k \in I$ for $1 \leq k \leq m$. For brevity, the brackets are omitted if an element has only one item. That is, element (x) is written as x. An item can occur at most once in an element of a sequence, but can occur multiple times in different elements of a sequence. The

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

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

- 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>

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

	<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>						

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

GSP Mining and Pruning

- 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

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]

SID	EID	Items
1	1	a
1	2	abc
1	3	ac
1	4	d
1	5	cf
2	1	ad
2	2	c
2	3	bc
2	4	ae
3	1	ef
3	2	ab
3	3	df
3	4	c
3	5	b
4	1	e
4	2	g
4	3	af
4	4	c
4	5	b
4	6	c

a		b		...
SID	EID	SID	EID	...
1	1	1	2	
1	2	2	3	
1	3	3	2	
2	1	3	5	
2	4	4	5	
3	2			
4	3			

ab			ba			...
SID	EID (a)	EID(b)	SID	EID (b)	EID(a)	...
1	1	2	1	2	3	
2	1	3	2	3	4	
3	2	5				
4	3	5				

aba				...
SID	EID (a)	EID(b)	EID(a)	...
1	1	2	3	
2	1	3	4	

https://pdfs.semanticscholar.org/39a0/80c17dec400a6f04af5fe5746dab3a5eb0dc.pdf

PrefixSpan: A Pattern-Growth Approach

- 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, …

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)>

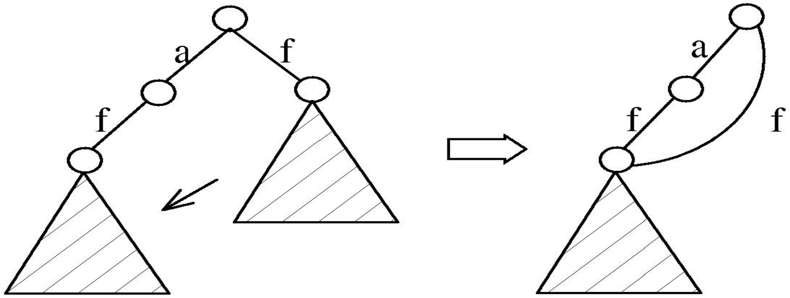
PrefixSpan (Prefix-projected Sequential pattern mining) Pei, et al. @TKDE'04

http://hanj.cs.illinois.edu/pdf/span01.pdf

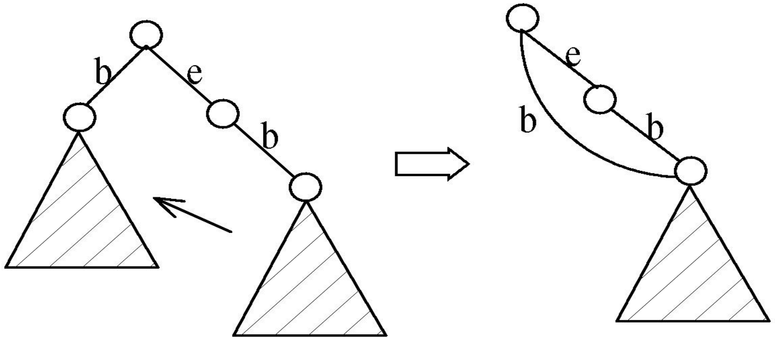
CloSpan: Mining Closed Sequential Patterns

- 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 $s \sqsupset s_1$, 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)

Backward subpattern pruning



Backward superpattern pruning



https://www.cs.ucsb.edu/~xyan/papers/cloSpan.pdf

Constraint-Based Sequential-Pattern Mining

- Share many similarities with constraint-based itemset mining
- Anti-monotonic: If S violates *c*, the super-sequences of S also violate *c*
  - sum(S.price) < 150; min(S.value) > 10
- Monotonic: If S satisfies *c*, the super-sequences of S also do so
  - element_count (S) > 5; S $\supseteq$ {PC, digital_camera}
- Data anti-monotonic: If a sequence s_1 with respect to S violates c_3, s_1 can be removed
  - c_3: sum(S.price) ≥ v
- Succinct: Enforce constraint c by explicitly manipulating data
  - S $\supseteq$ {i-phone, MacAir}
- Convertible: Projection based on the sorted value not sequence order
  - value_avg(S) < 25; profit_sum (S) > 160
  - max(S)/avg(S) < 2; median(S) – min(S) > 5

Timing-Based Constraints in Seq.-Pattern Mining

- Order constraint: Some items must happen before the other
  - {algebra, geometry} → {calculus} (where "→" indicates ordering)
  - Anti-monotonic: Constraint-violating sub-patterns pruned
- Min-gap/max-gap constraint: Confines two elements in a pattern
  - E.g., mingap = 1, maxgap = 4
  - Succinct: Enforced directly during pattern growth
- Max-span constraint: Maximum allowed time difference between the 1st and the last elements in the pattern
  - E.g., maxspan (S) = 60 (days)
  - Succinct: Enforced directly when the 1st element is determined
- Window size constraint: Events in an element do not have to occur at the same time: Enforce max allowed time difference
  - E.g., window-size = 2: Various ways to merge events into elements

Episodes and Episode Pattern Mining

- Episodes and regular expressions: Alternative to seq. patterns
  - Serial episodes: $A \rightarrow B$
  - Parallel episodes: $A \mid B$ Indicating partial order relationships
  - Regular expressions: $(A|B)C^*(D \rightarrow E)$
- Methods for episode pattern mining
  - Variations of Apriori/GSP-like algorithms
  - Projection-based pattern growth
    - Q_1: Can you work out the details?
  - Q_2: What are the differences between mining episodes and constraint-based pattern mining?

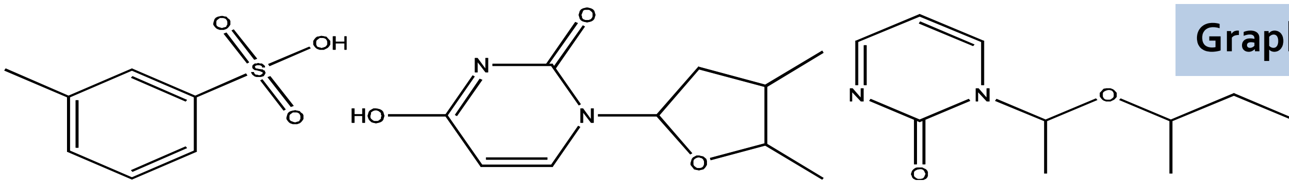
Advanced Frequent Pattern Mining

- Mining Diverse Patterns
- Constraint-Based Frequent Pattern Mining
- Mining High-Dimensional Data and Colossal Patterns
- Sequential Pattern Mining
- **Graph Pattern Mining**

Frequent (Sub)Graph Patterns

- Given a labeled graph dataset $D = \{G_1, G_2, \ldots, G_n\}$, the supporting graph set of a subgraph g is $D_g = \{G_i \mid g \subseteq G_i, G_i \in D\}$.
  - 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

Graph Dataset

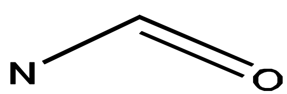


(A) (B) (C)

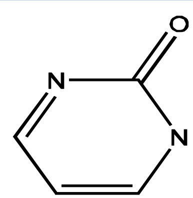
min_sup = 2

Frequent Graph Patterns

(1)



(2)



Applications of Graph Pattern Mining

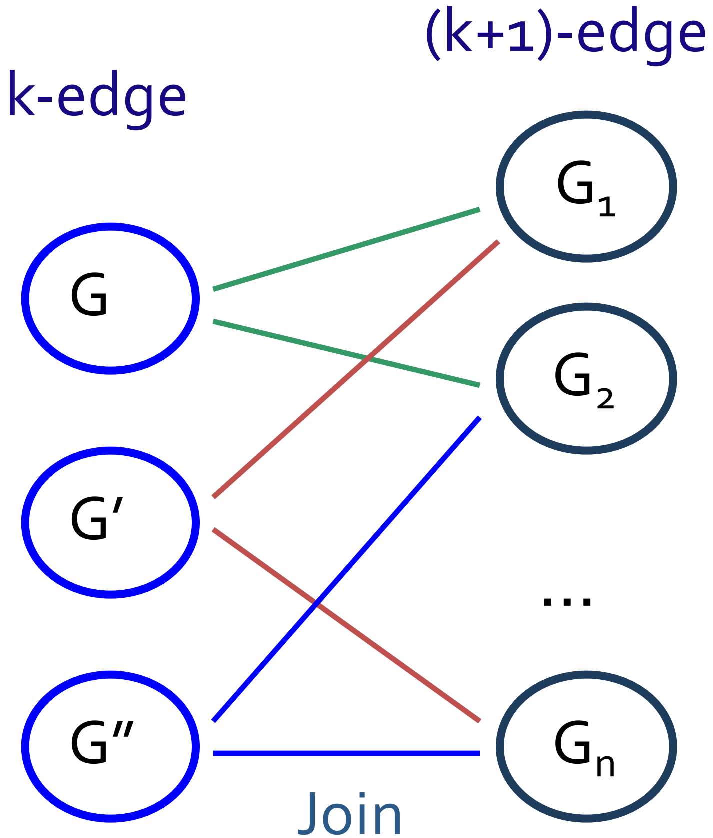
- Bioinformatics
  - Gene networks, protein interactions, metabolic pathways
- Chem-informatics: Mining chemical compound structures
- Social networks, web communities, tweets, …
- Cell phone networks, computer networks, …
- Web graphs, XML structures, semantic Web, information networks
- Software engineering: program execution flow analysis
- Building blocks for graph classification, clustering, compression, comparison, and correlation analysis
- Graph indexing and graph similarity search

Graph Pattern Mining Algorithms: Different Methodologies

- Generation of candidate subgraphs
  - Apriori vs. pattern growth (e.g., FSG vs. gSpan)
- Search order
  - Breadth vs. depth
- Elimination of duplicate subgraphs
  - Passive vs. active (e.g., gSpan (Yan&Han'02))
- Support calculation
  - Store embeddings (e.g., GASTON (Nijssen&Kok'04, FFSM (Huan, et al.'03), MoFa (Borgelt and Berthold ICDM'02))
- Order of pattern discovery
  - Path → tree → graph (e.g., GASTON (Nijssen&Kok'04))

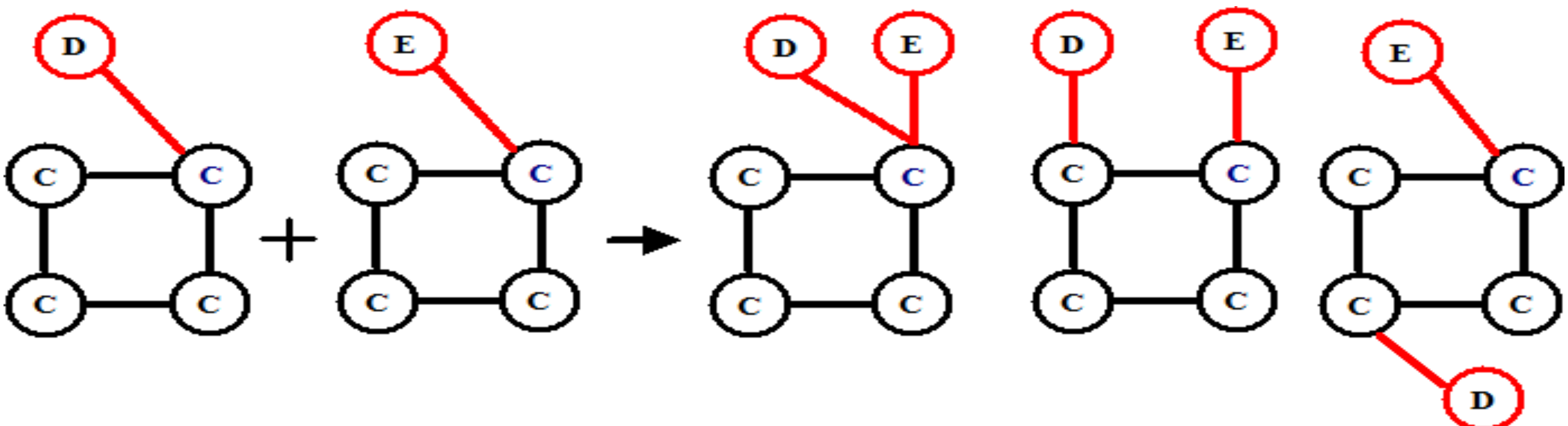
Apriori-Based Approach

- 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



Candidate Generation: Vertex Growing vs. Edge Growing

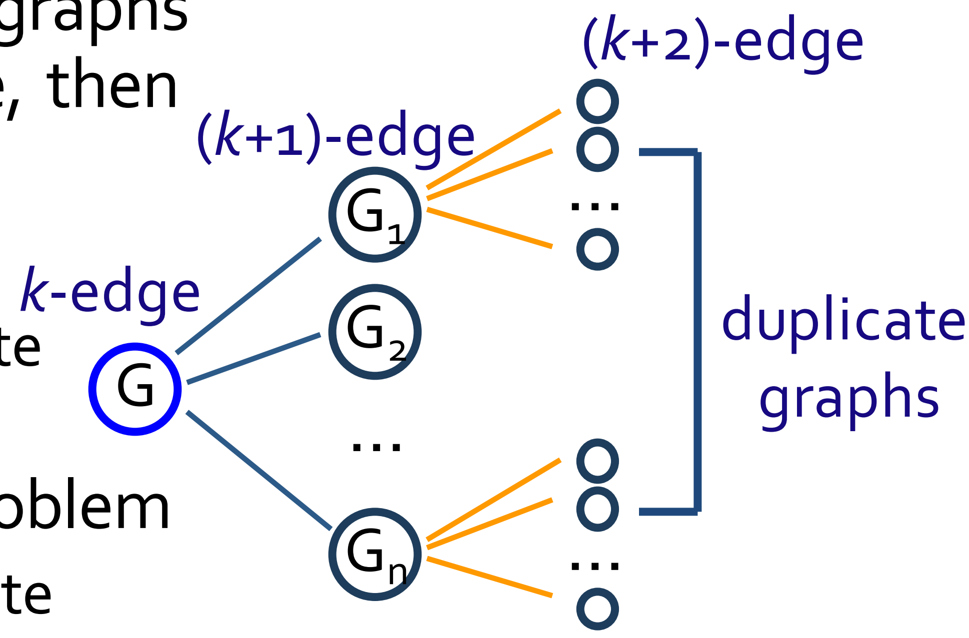
- Methodology: breadth-search, Apriori joining two size-k graphs
  - Many possibilities at generating size-(k+1) candidate graphs



- Generating new graphs with one more vertex
  - AGM (Inokuchi, et al., PKDD'00)
- Generating new graphs with one more edge
  - FSG (Kuramochi and Karypis, ICDM'01)
- Performance shows via edge growing is more efficient

Pattern-Growth Approach

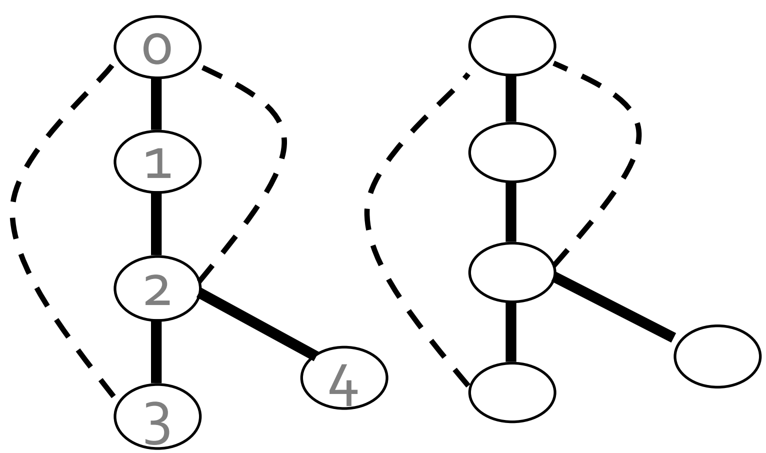
- Depth-first growth of subgraphs from k-edge to $(k+1)$-edge, then $(k+2)$-edge subgraphs
- Major challenge
  – Generating many duplicate subgraphs
- Major idea to solve the problem
  – Define an order to generate subgraphs
  – DFS spanning tree: Flatten a graph into a sequence using depth-first search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



http://cs.ucsb.edu/~xyan/papers/gSpan-short.pdf

gSPAN: Graph Pattern Growth in Order

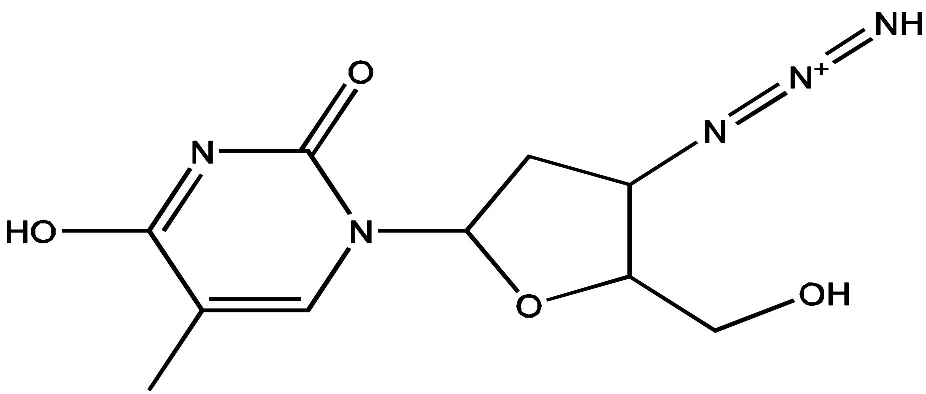
- **Right-most path extension** in subgraph pattern growth
  - Right-most path: The path from root to the right-most leaf (choose the vertex w. the smallest index at each step)
  - Reduce generation of duplicate subgraphs
- **Completeness:** The Enumeration of graphs using right-most path extension is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



e_0: (0,1)
e_1: (1,2)
e_2: (2,0)
e_3: (2,3)
e_4: (3,0)
e_5: (2,4)

Why Mining Closed Graph Patterns?

- Challenge: An **n**-edge frequent graph may have 2^n subgraphs
- Motivation: Explore *closed frequent subgraphs* to handle graph pattern explosion problem
- A frequent graph G is *closed* if there exists no supergraph of G that carries the same support as G

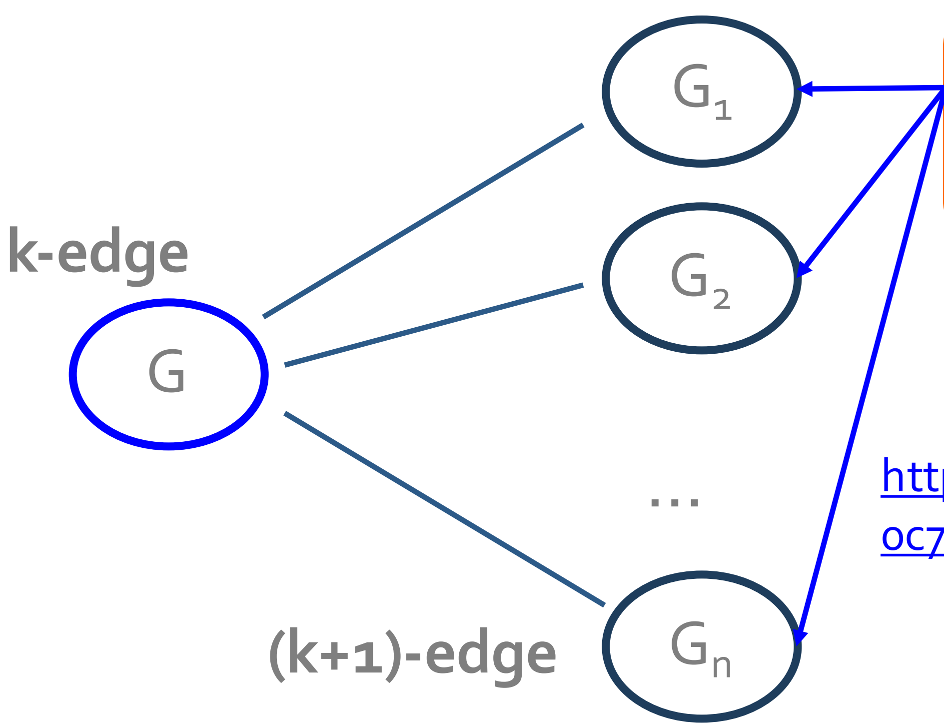


If this subgraph is *closed* in the graph dataset, it implies that none of its frequent super-graphs carries the same support

- *Lossless compression:* Does not contain non-closed graphs, but still ensures that the mining result is complete
- Algorithm CloseGraph: Mines closed graph patterns directly

CloseGraph: Directly Mining Closed Graph Patterns

- CloseGraph: Mining closed graph patterns by extending gSpan
- Suppose G and G_1 are frequent, and G is a subgraph of G_1
- If **in any part of the graph in the dataset where G occurs, G_1 also occurs**, then we need not grow G (except some special, subtle cases), since none of G's children will be closed except those of G_1



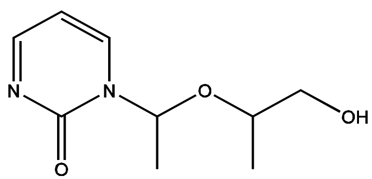
At what condition can we stop searching their children, i.e., early termination?

https://pdfs.semanticscholar.org/a1c1/5e63690c774b725fc91dcc77a629d01c3733.pdf

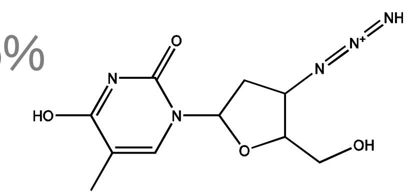
Experiment and Performance Comparison

- The AIDS antiviral screen compound dataset from NCI/NIH
- The dataset contains 43,905 chemical compounds
- Discovered Patterns: The smaller minimum support, the bigger and more interesting subgraph patterns discovered

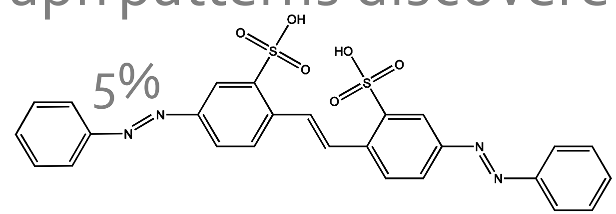
20%



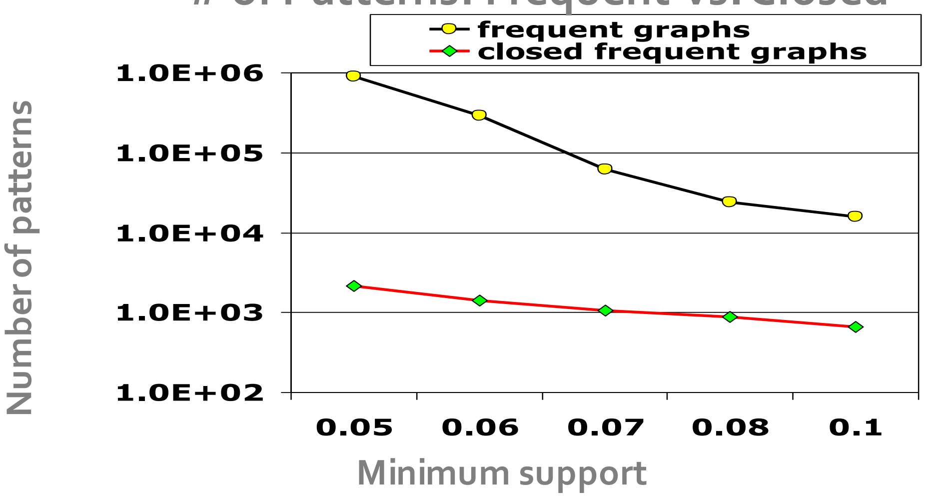
10%



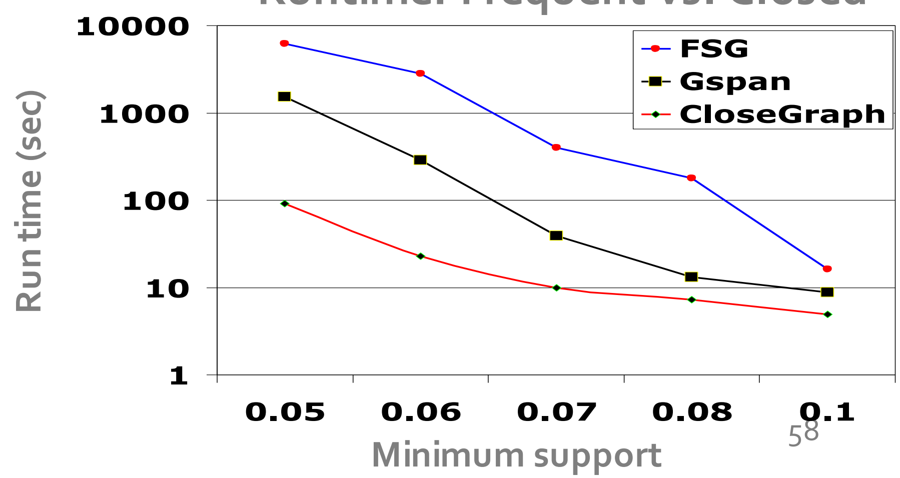
5%



# of Patterns: Frequent vs. Closed



Runtime: Frequent vs. Closed



References: Mining Diverse Patterns

- R. Srikant and R. Agrawal, “Mining generalized association rules”, VLDB'95
- Y. Aumann and Y. Lindell, “A Statistical Theory for Quantitative Association Rules”, KDD'99
- K. Wang, Y. He, J. Han, “Pushing Support Constraints Into Association Rules Mining”, IEEE Trans. Knowledge and Data Eng. 15(3): 642-658, 2003
- D. Xin, J. Han, X. Yan and H. Cheng, "On Compressing Frequent Patterns", Knowledge and Data Engineering, 60(1): 5-29, 2007
- D. Xin, H. Cheng, X. Yan, and J. Han, "Extracting Redundancy-Aware Top-K Patterns", KDD'06
- J. Han, H. Cheng, D. Xin, and X. Yan, "Frequent Pattern Mining: Current Status and Future Directions", Data Mining and Knowledge Discovery, 15(1): 55-86, 2007
- F. Zhu, X. Yan, J. Han, P. S. Yu, and H. Cheng, “Mining Colossal Frequent Patterns by Core Pattern Fusion”, ICDE'07

References: Constraint-Based Frequent Pattern Mining

- R. Srikant, Q. Vu, and R. Agrawal, “Mining association rules with item constraints”, KDD'97
- R. Ng, L.V.S. Lakshmanan, J. Han & A. Pang, “Exploratory mining and pruning optimizations of constrained association rules”, SIGMOD’98
- G. Grahne, L. Lakshmanan, and X. Wang, “Efficient mining of constrained correlated sets”, ICDE'00
- J. Pei, J. Han, and L. V. S. Lakshmanan, “Mining Frequent Itemsets with Convertible Constraints”, ICDE'01
- J. Pei, J. Han, and W. Wang, “Mining Sequential Patterns with Constraints in Large Databases”, CIKM'02
- F. Bonchi, F. Giannotti, A. Mazzanti, and D. Pedreschi, “ExAnte: Anticipated Data Reduction in Constrained Pattern Mining”, PKDD'03
- F. Zhu, X. Yan, J. Han, and P. S. Yu, “gPrune: A Constraint Pushing Framework for Graph Pattern Mining”, PAKDD'07

References: Sequential Pattern Mining

- R. Srikant and R. Agrawal, “Mining sequential patterns: Generalizations and performance improvements”, EDBT’96
- M. Zaki, “SPADE: An Efficient Algorithm for Mining Frequent Sequences”, Machine Learning, 2001
- J. Pei, J. Han, B. Mortazavi-Asl, J. Wang, H. Pinto, Q. Chen, U. Dayal, and M.-C. Hsu, "Mining Sequential Patterns by Pattern-Growth: The PrefixSpan Approach", IEEE TKDE, 16(10), 2004
- X. Yan, J. Han, and R. Afshar, “CloSpan: Mining Closed Sequential Patterns in Large Datasets”, SDM'03
- J. Pei, J. Han, and W. Wang, "Constraint-based sequential pattern mining: the pattern-growth methods", J. Int. Inf. Sys., 28(2), 2007
- M. N. Garofalakis, R. Rastogi, K. Shim: Mining Sequential Patterns with Regular Expression Constraints. IEEE Trans. Knowl. Data Eng. 14(3), 2002
- H. Mannila, H. Toivonen, and A. I. Verkamo, “Discovery of frequent episodes in event sequences”, Data Mining and Knowledge Discovery, 1997

References: Graph Pattern Mining

- C. Borgelt and M. R. Berthold, Mining molecular fragments: Finding relevant substructures of molecules, ICDM'02
- J. Huan, W. Wang, and J. Prins. Efficient mining of frequent subgraph in the presence of isomorphism, ICDM'03
- A. Inokuchi, T. Washio, and H. Motoda. An apriori-based algorithm for mining frequent substructures from graph data, PKDD'00
- M. Kuramochi and G. Karypis. Frequent subgraph discovery, ICDM'01
- S. Nijssen and J. Kok. A Quickstart in Frequent Structure Mining can Make a Difference. KDD'04
- N. Vanetik, E. Gudes, and S. E. Shimony. Computing frequent graph patterns from semistructured data, ICDM'02
- X. Yan and J. Han, gSpan: Graph-Based Substructure Pattern Mining, ICDM'02
- X. Yan and J. Han, CloseGraph: Mining Closed Frequent Graph Patterns, KDD'03
- X. Yan, P. S. Yu, J. Han, Graph Indexing: A Frequent Structure-based Approach, SIGMOD'04
- X. Yan, P. S. Yu, and J. Han, Substructure Similarity Search in Graph Databases, SIGMOD'05