CS 412 Intro. to Data Mining

Chapter 6 : Advanced Pattern Mining

Arindam Banerjee, Computer Science, UIUC, Fall 2022

Chapter 6 : Advanced Frequent Pattern Mining

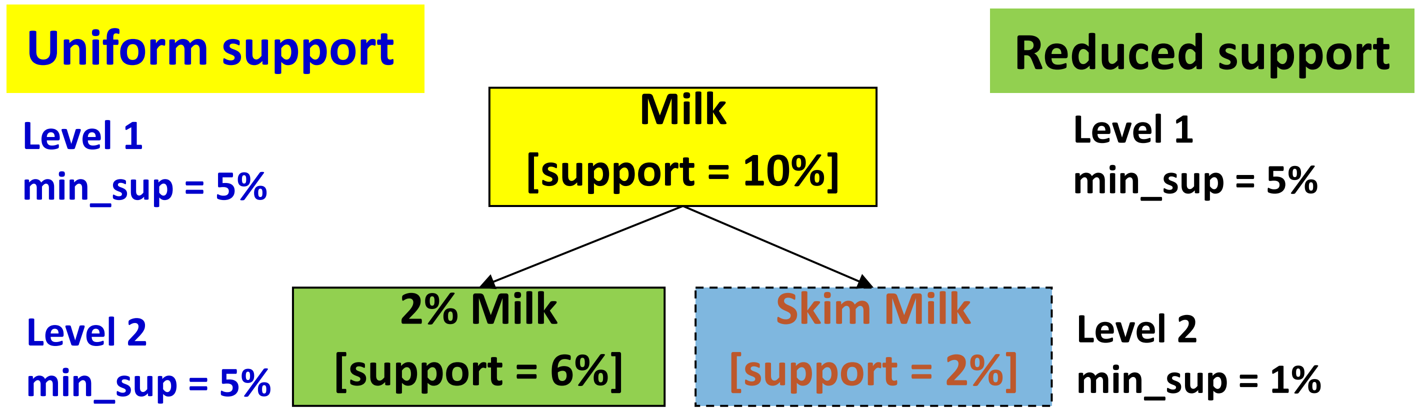
- Mining Diverse Patterns
- Constraint-Based Frequent Pattern Mining
- Sequential Pattern Mining
- Graph Pattern Mining
- Summary

Mining Diverse Patterns

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

Mining Multiple-Level Frequent Patterns

- Min-support thresholds for hierarchy items
  - **Uniform** min-support across multiple levels (reasonable?)



  - **Level-reduced** min-support: Items at the lower level are expected to have lower support
- Efficient mining: *Shared* multi-level mining
  - Use the lowest min-support to pass down the set of candidates

Redundancy Filtering at Mining Multi-Level Associations

- Redundancy filtering: redundant due to “ancestor” relationships
  - milk $\Rightarrow$ wheat bread [support = 8%, confidence = 70%]   (1)
  - 2% milk $\Rightarrow$ wheat bread [support = 2%, confidence = 72%] (2)
    - Suppose the 2% milk sold is about ¼ of milk sold
      - (2) should be able to be “derived” from (1)

Redundancy Filtering at Mining Multi-Level Associations

- 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

- **Same** min-support threshold **for all** so far
- Diamonds, watches: valuable but **less** frequent
- One Method: Use **group-based** "individualized" min-support
  - E.g., {diamond, watch}: 0.05%; {bread, milk}: 5%; …
  - How to mine such rules efficiently?
    - Existing scalable mining algorithms can be easily extended to cover such cases

Mining Multi-Dimensional Associations

- **Single**-dimensional rules (e.g., items are all in "product" dimension)
  - buys(X, "milk") $\Rightarrow$ buys(X, "bread")
- **Multi**-dimensional rules (i.e., items in $\geq$ 2 dimensions or predicates)
  - Inter-dimension association rules (*no repeated predicates*)
    - age(X, "18-25") $\wedge$ occupation(X, "student") $\Rightarrow$ buys(X, "coke")
  - Hybrid-dimension association rules (*repeated predicates*)
    - age(X, "18-25") $\wedge$ buys(X, "popcorn") $\Rightarrow$ 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
  - E.g.: Numerical attributes: age and salary
- Methods
  - **Static discretization** based on predefined concept hierarchies
    - Discretization on each dimension with hierarchy
      - age: {0-10, 10-20, …, 90-100} → {young, mid-aged, old}
  - **Dynamic discretization** based on data distribution
  - **Clustering**: Distance-based association
    - First one-dimensional clustering, then association
  - **Deviation analysis**:
    - Gender = female $\Rightarrow$ Wage: mean=$7/hr (overall mean = $9)

Mining Extraordinary Phenomena in Quantitative Association Mining

- Mining extraordinary (i.e., interesting) phenomena
  - E.g.: Gender = female $\Rightarrow$ 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
  - E.g.: (Gender = female) ^ (South = yes) $\Rightarrow$ mean wage = $6.3/hr
- Rule condition can be categorical or numerical (quantitative rules)
  - E.g.: Education in [14-18] (yrs) $\Rightarrow$ mean wage = $11.64/hr

Rare 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

- Negative patterns
  - Negatively correlated: Unlikely to happen together
  - Ex.: Since it is unlikely that the same customer buys both a Ford Expedition (SUV) and a Ford Fusion (hybrid), buying a Ford Expedition and buying a Ford Fusion are likely negatively correlated patterns
  - How to define negative patterns?
- A support-based definition of negative correlated patterns
  - If itemsets A and B are both frequent but rarely occur together, i.e., $sup(A \cup B) \ll sup(A) \times sup(B)$

Does this remind you the definition of *lift*?

Defining Negative Correlated Patterns

Is this a good definition for large transaction datasets?

- Ex.: Suppose a dealership sells vehicles:
  - Types A and B 100 sold times each, but only one transaction contained both A and B
  - Consider total 200 transactions (199 contain A or B), 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 Kulczynski measure-based definition
  - If itemsets A and B are frequent but $(s(A \cup B)/s(A) + s(A \cup B)/s(B))/2 < \epsilon$, then A and B are negatively correlated
- For the same car buying problem:
  - Does not matter if there are in total 200 or 10^5 transactions
    - If $\epsilon = 0.01$, we have

      $(s(A \cup B)/s(A) + s(A \cup B)/s(B))/2 = (0.01 + 0.01)/2 < \epsilon$

negative pattern threshold

Mining Compressed Patterns

Pat-ID	Item-Sets	Support
P1	{38,16,18,12}	205,227
P2	{38,16,18,12,17}	205,211
P3	{39,38,16,18,12,17}	101,758
P4	{39,16,18,12,17}	161,563
P5	{39,16,18,12}	161,576

- Closed patterns
  - P1, P2, P3, P4, P5
  - Emphasizes too much on support
- Max-patterns
  - P3: information loss
- Desired output (a good balance):
  - P2, P3, P4

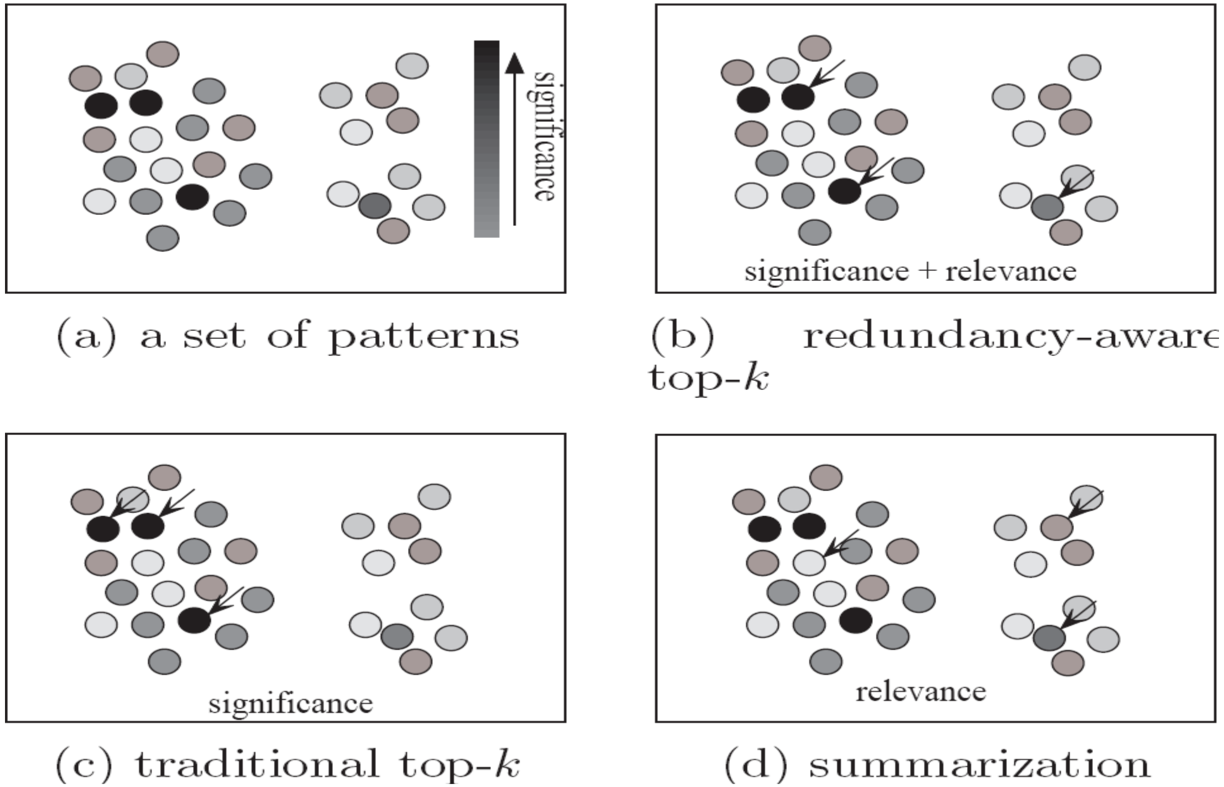
- Why mining compressed patterns? Too many scattered patterns but not so meaningful
- Pattern distance measure

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

- 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

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

Chapter 6 : Advanced Frequent Pattern Mining

- Mining Diverse Patterns
- Constraint-Based Frequent Pattern Mining
- Sequential Pattern Mining
- Graph Pattern Mining
- Summary

Constraint-Based Pattern Mining

- Why Constraint-Based Mining?
- Different Kinds of Constraints: Different Pruning Strategies
- Constrained Mining with Pattern Anti-Monotonicity
- Constrained Mining with Pattern Monotonicity
- Constrained Mining with Convertible Constraints
- Constrained Mining with Data Anti-Monotonicity
- Constrained Mining with Succinct Constraints
- Handling Multiple Constraints

Why Constraint-Based Mining?

- Pattern mining in practice: Often a user-guided, interactive process
  - User directs what to be mined using a data mining query language (or a graphical user interface), specifying various kinds of constraints
- What is constraint-based mining?
  - Mine together with user-provided constraints
- Why constraint-based mining?
  - User flexibility: User provides constraints on what to be mined
  - Optimization: System explores such constraints for mining efficiency
    - E.g., Push constraints deeply into the mining process

Various Kinds of User-Specified Constraints in Data Mining

- Knowledge type constraint—Specifying what kinds of knowledge to mine
  - E.g.: Classification, association, clustering, outlier finding, …
- Data constraint—using SQL-like queries
  - E.g.: Find products sold together in NY stores this year
- Dimension/level constraint—similar to projection in relational database
  - E.g.: In relevance to region, price, brand, customer category
- Interestingness constraint—various kinds of thresholds
  - E.g.: Strong rules: min_sup $\geq$ 0.02, min_conf $\geq$ 0.6, min_correlation $\geq$ 0.7
- Rule (or pattern) constraint ← The focus of this study
  - E.g.: Small sales (price < $10) triggers big sales (sum > $200)

Pattern Space Pruning with Pattern Anti-Monotonicity

Item	Price	Profit
a	100	40
b	40	0
c	150	−20
d	35	−15
e	55	−30
f	45	−10
g	80	20
h	10	5

Note: item.price > 0
Profit can be negative

- A 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
- E.g. 1: c_1: *sum*(*S.price*) $\leq$ 160 is anti-monotone
  - Sum grows as you add more items
  - Itemset *abc* violates c_1
- E.g. 2: c_2: *range*(S.profit) $\leq$ 15 is anti-monotone
  - Itemset *ab* violates c_2 (*range*(ab) = 40)
  - So does every superset of *ab*

Pattern Space Pruning with Pattern Anti-Monotonicity

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

Item	Price	Profit
a	100	40
b	40	0
c	150	−20
d	35	−15
e	55	−30
f	45	−10
g	80	20
h	10	5

- E.g. 3. c_3: $sum(S.Price) \geq 160$ is not anti-monotone
  - *ab* violates the constrint, but super-sets of *ab* need not violate
  - *abc*, *abd*, etc., does not violate the constraint
  - Cannot be used for pruning super-sets
- E.g. 4. Is c_4: $support(S) \geq \sigma$ anti-monotone?
  - Yes! Apriori pruning is essentially pruning with an anti-monotonic constraint!

Pattern Monotonicity and Its Roles

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

Item	Price	Profit
a	100	40
b	40	0
c	150	−20
d	35	−15
e	55	−30
f	45	−10
g	80	20
h	10	5

- 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
  - Not as beneficial as anti-monotone
- E.g. 1: c_1: *sum*(*S.Price*) $\geq$ 160 is monotone
- E.g. 2: c_2: *min*(*S.Price*) $\leq$ 50 is monotone
- E.g. 3: c_3: range(S.profit) $\geq$ 15 is monotone
  - Itemset *ab* satisfies c_3
  - So does every superset of *ab*

Apriori for Pattern Anti-Monotone Constraint

Item	Price
1	1
2	2
3	3
4	4
5	5

Database D

TID	Items
10	1 3 4
20	2 3 5
30	1 2 3 5
40	2 5

Scan D

C_1

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

F_1

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

Can be chopped early

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~~

F_2

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

C_3

itemset
~~{2 3 5}~~

Scan D

F_3

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

Min_sup=2

Constraint:

Sum{S.price} < 5

Convertible Constraints: Ordering Data in Transactions

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

min_sup = 2

Item	Price	Profit
a	100	40
b	40	0
c	150	−20
d	35	−15
e	55	−30
f	45	−5
g	80	30
h	10	5

- Convert tough constraints into (anti-)monotone by proper ordering of items in transactions
- Examine c_1: avg(*S*.profit) > 35
  - Order items in (profit) value-descending order
    - *<a, g, h, b, f, d, c, e>*
  - An itemset *ag* violates c_1 (avg(ag) = 35)
    - So does *ag** (i.e., *ag-projected DB)*
    - C_1: anti-monotone if patterns grow in the right order!

Can item-reordering work for Apriori?

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

min_sup = 2

Item	Price	Profit
a	100	40
b	40	0
c	150	−20
d	35	−15
e	55	−30
f	45	−5
g	80	30
h	10	5

constraint: avg(S.profit) > 20

Scan D →

F_1

itemset	sup.
{a}	3
{f}	4
{g}	3
{d}	3
…	…

F_2

itemset	sup.
~~{af}~~	~~3~~
~~{fg}~~	~~4~~
{ag}	2
~~{ad}~~	~~2~~
…	…

Chopped too early

- avg(f*g*) = 12.5 < 20, avg(af) = 17.5 < 20, avg(ag) = 35 > 20
- But *avg*(*agf*) = 21.7 > 20
- Apriori will not generate "agf" as a candidate

Data Space Pruning with Data Anti-Monotonicity

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

Item	Price	Profit
a	100	40
b	40	0
c	150	−20
d	35	−15
e	55	−30
f	45	−10
g	80	20
h	10	5

- A constraint c is *data anti-monotone*: In the mining process, if a data entry *t* cannot contribute to a pattern *p* satisfying *c*, *t* cannot contribute to *p*'s superset either
  - Data space pruning: Data entry *t* can be pruned

- E.g. 1: c_1: $sum(S.Profit) \geq v$ is data anti-monotone
  - Let constraint c_1 be: $sum(S.Profit) \geq 25$
    - T_{30}: {b, c, d, f, h} can be removed since none of their combinations can make an S whose sum of the profit is ≥ 25

Data Space Pruning with Data Anti-Monotonicity

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

Item	Price	Profit
a	100	40
b	40	0
c	150	−20
d	35	−15
e	55	−30
f	45	−10
g	80	20
h	10	5

- A constraint c is *data anti-monotone*: In the mining process, if a data entry *t* cannot contribute to a pattern *p* satisfying *c*, *t* cannot contribute to *p*'s superset either
  - Data space pruning: Data entry *t* can be pruned
- E.g. 2: c_2: $min(S.Price) \leq v$ is data anti-monotone
  - Consider $v = 5$ but every item in a transaction, say T_{50}, has a price higher than 10
- E.g. 3: c_3: $range(S.Profit) > 25$ is data anti-monotone
  - Transaction {c,d,e,f} has a range of 20

Data Space Pruning Should Be Explored Recursively

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

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

- Example. c_3: *range*(*S.Profit*) > *25*
  - We check b's projected database
    - T_{10} satisfies c_3
    - But item "a" is infrequent (sup = 1)

b's-proj. DB

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

- After removing "a (40)" from T_{10}
  - T_{10} cannot satisfy c_3 any more
    - Since "b (0)" and "c (−20), d (−15), f (−10), h
  - By removing T_{10}, we can also prune "h"

b's-proj. DB

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

Data Space Pruning Should Be Explored Recursively

b's-proj. DB

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

Recursive Data Pruning

b's FP-tree

single branch: cdfg: 2

Constraint:
range{S.profit} > 25

Only a single branch "cdfg: 2" to be mined in b's projected DB

- Note: c_3 prunes T_{10} effectively only after "a" is pruned (by min-sup) in b's projected DB

Succinctness: Pruning Both Data and Pattern Spaces

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

Constrained FP-Growth: Push a Succinct Constraint Deep

Item	Price
1	1
2	2
3	3
4	4
5	5

TID	Items
10	1 3 4
20	2 3 5
30	1 2 3 5
40	2 5

Remove infrequent length 1

TID	Items
10	1 3
20	2 3 5
30	1 2 3 5
40	2 5

Min_sup=2

Constraint:

min{S.price} <= 2

No Need to project on 3 or 5

1-Projected DB

TID	Items
10	3
30	2 3 5

2-Projected DB

TID	Items
20	3 5
30	1 3 5
40	5

Commonly Used Pattern Pruning Constraints

Table 6.3: Characterization of Commonly Used Pattern Pruning Constraints

Constraint	*Antimonotonic*	*Monotonic*	*Succinct*
$v \in S$	no	yes	yes
$S \supseteq V$	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	no
$count(S) \geq v$	no	yes	no
$sum(S) \leq v$ $(\forall a \in S,\ a \geq 0)$	yes	no	no
$sum(S) \geq v$ $(\forall a \in S,\ a \geq 0)$	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
$all\_confidence(S) \geq \xi$	yes	no	no
$all\_confidence(S) \leq \xi$	no	yes	no

Different Kinds of Constraints Lead to Different Pruning Strategies

- In summary, constraints can be categorized as pattern space pruning constraints vs. data space pruning constraints

Pattern space pruning constraints	Data 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 ▪ Convertible: *c* can be converted to monotonic or anti-monotonic if items can be properly ordered in processing ▪ Succinct: If the constraint *c* can be enforced by directly manipulating the data	▪ 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

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 conflict ordering between c_1 and c_2
    - Try to sort data and enforce *one constraint* first (which one?)
    - Then enforce the other constraint when mining the projected databases
- E.g. c_1: avg(*S*.profit) **>** 20, and c_2: avg(S.price) < 50
  - Assume c_1 has more pruning power
    - Sort in profit descending order and use c_1 first
  - For each project DB, sort transactions in price ascending order and use c_2 during mining

Chapter 6 : Advanced Frequent Pattern Mining

- Mining Diverse Patterns
- Constraint-Based Frequent Pattern Mining
- Sequential Pattern Mining
- Graph Pattern Mining
- Summary

Sequential Pattern Mining

- Sequential Pattern and Sequential Pattern Mining
- GSP: Apriori-Based Sequential Pattern Mining
- SPADE: Sequential Pattern Mining in Vertical Data Format
- PrefixSpan: Sequential Pattern Mining by Pattern-Growth
- CloSpan: Mining Closed Sequential Patterns
- Constraint-Based Sequential-Pattern Mining

Sequential Pattern Mining

- What kind of patterns are sequential?
- Sequential – The order really matters. You can not swap two items in a sequence and have the same sequence.
- Examples: English language is sequential : Subject -> Verb -> Object, amino acid sequences, etc.

- Other points:
  - For Sequential Pattern Mining, the time at which the items occur is **not** considered.
  - Time Series Analysis does take into account the time at which an item occurred.

Sequential Pattern Examples

- Application of Sequential pattern Mining
  - **Customer shopping** → Purchase a laptop first, then a digital camera, and then a smartphone.
  - **Medical treatments** → Go to the doctor, get drugs, doctor monitors progress, doctor reacts accordingly -> more/less drugs
  - **Natural disasters** -> Before the disaster, during the disaster, after the disaster.
  - **Scientific Experiments** → Step 1, Step 2, Step 3.
  - **Stocks Markets** → Stocks go up and down together.
  - **Biological sequences, DNA /Protein**→ If you change the order of proteins, it is a different gene.

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)

element (unordered within "(..)")

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 **sequence database**

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

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(bc)dc> is a ***subsequence*** of <a(abc)(ac)d(cf)>

- Given support threshold *min_sup* = 2, <(ab)c> *is a* *sequential pattern*

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>

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 (to be covered in the constraint mining section)

GSP: Apriori-Based Sequential Pattern Mining

- Initial candidates: All 8-singleton sequences
  - <a>, <b>, <c>, <d>, <e>, <f>, <g>, <h>
- Scan DB once, count support for each candidate

x

min_sup = 2

SID	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: Apriori-Based Sequential Pattern Mining

- Example: 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

singleton * singleton – *Total:* (6 * 6)

	<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>

Sets (unordered) – *Total*: (6*5) / 2

	<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>						

Apriori Pruning

- w/o pruning *(includes g and h):* 8*8 + 8*7/2 = 92 length-2 candidates
- w/ pruning: 6*6 + 6*5/2 = 51 length-2 candidates

GSP Mining and Pruning

		length
5th scan: 1 cand. 1 length-5 seq. pat.	<(bd)cba>	5
4th scan: 8 cand. 7 length-4 seq. pat.	<abba> <(bd)bc> …	4
3rd scan: 46 cand. 20 length-3 seq. pat. 20 cand. not in DB at all	<abb> <aab> <aba> <baa> <bab> …	3
2nd scan: 51 cand. 19 length-2 seq. pat. 10 cand. not in DB at all	<aa> <ab> … <af> <ba> <bb> … <ff> <(ab)> … <(ef)>	2
1st scan: 8 cand. 6 length-1 seq. pat.	<a> <b> <c> <d> <e> <f> ~~<g> <h>~~	1

6*6 + 6*5/2 = 51

- Remove
  - Candidates not in DB
  - Candidates < min_sup

min_sup = 2

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

GSP Mining and Pruning

- Repeat, starting at k=1 until k<=length
  - 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

GSP (Generalized Sequential Patterns): Srikant & Agrawal @ EDBT'96)
-NOTE: Same team which developed Apriori

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				

EID (b) < EID (a): Corresponds to: <a(abc)(ac)d(cf)>

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

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>

min_sup = 2

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

"_" is placeholder for prefix

- 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>

min_sup = 2

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

prefix <a>

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

prefix <b>

<b>-projected DB
<(_c)(ac)d(cf)>
<(_c)(ae)>
<(df)cb>
<c>

… …

prefix <c>, …, <f>

…

prefix <aa>

prefix <af>

<aa>-projected DB ..

<af>-projected DB

Major strength of PrefixSpan:

- No candidate subseqs. to be generated
- Projected DBs keep shrinking

PrefixSpan: Key Steps

1. Let $\{\langle x_1\rangle, \langle x_2\rangle, \ldots, \langle x_n\rangle\}$ be the complete set of length-1 sequential patterns in a sequence database, S. The complete set of sequential patterns in S can be partitioned into n disjoint subsets. The i^{th} subset $(1 \leq i \leq n)$ is the set of sequential patterns with prefix $\langle x_i\rangle$.

2. Let α be a length-l sequential pattern and $\{\beta_1, \beta_2, \ldots, \beta_m\}$ be the set of all length-$(l+1)$ sequential patterns with prefix α. The complete set of sequential patterns with prefix α, except for α itself, can be partitioned into m disjoint subsets. The j^{th} subset $(1 \leq j \leq m)$ is the set of sequential patterns prefixed with β_j.

PrefixSpan: Example

Sequence_ID	*Sequence*
1	$\langle a(abc)(ac)d(cf)\rangle$
2	$\langle (ad)c(bc)(ae)\rangle$
3	$\langle (ef)(ab)(df)cb\rangle$
4	$\langle eg(af)cbc\rangle$

1. Find length-1 patterns
<a>: 4, <b>: 4, <c>: 4, <d>: 3, <e>: 3, <f>: 3

2. Partition the search space, based on prefix

3. Find subsets of patterns: <a>-projected db, frequent
<a>: 2, <b>: 4, <_b>: 2, <c>: 4, <d>: 2, <f>: 2

4. All patterns found recursively

prefix	projected database
$\langle a\rangle$	$\langle (abc)(ac)d(cf)\rangle$, $\langle (\_d)c(bc)(ae)\rangle$, $\langle (\_b)(df)cb\rangle$, $\langle (\_f)cbc\rangle$
$\langle b\rangle$	$\langle (\_c)(ac)d(cf)\rangle$, $\langle (\_c)(ae)\rangle$, $\langle (df)cb\rangle$, $\langle c\rangle$
$\langle c\rangle$	$\langle (ac)d(cf)\rangle$, $\langle (bc)(ae)\rangle$, $\langle b\rangle$, $\langle bc\rangle$
$\langle d\rangle$	$\langle (cf)\rangle$, $\langle c(bc)(ae)\rangle$, $\langle (\_f)cb\rangle$
$\langle e\rangle$	$\langle (\_f)(ab)(df)cb\rangle$, $\langle (af)cbc\rangle$
$\langle f\rangle$	$\langle (ab)(df)cb\rangle$, $\langle cbc\rangle$

PrefixSpan: Example (Contd.)

Sequence_ID	*Sequence*
1	$\langle a(abc)(ac)d(cf)\rangle$
2	$\langle (ad)c(bc)(ae)\rangle$
3	$\langle (ef)(ab)(df)cb\rangle$
4	$\langle eg(af)cbc\rangle$

1. Find length-1 patterns
<a>: 4, <b>: 4, <c>: 4, <d>: 3, <e>: 3, <f>: 3

2. Partition the search space, based on prefix

3. Find subsets of patterns: <a>-projected db, frequent <a>: 2, <b>: 4, <_b>: 2, <c>: 4, <d>: 2, <f>: 2

4. All patterns found recursively

Projected databases:
<aa> : <(_bc)(ac)d(cf)>, <(_e)>
not frequent, stop

<ab> : <(_c)(ac)d(cf)>, <(_c)(ae)>, <c>
frequent: <(_c)>, <a>, <c>
<a(bc)> : <(ac)d(cf)>, <(ae)>
frequent: <a>
sequential patterns: <a(bc)>, <aba>, <abc>,
<a(bc)a>

<(ab)> : <(_c)(ac)d(cf)>, <(df)cb>
frequent: <c>, <d>, <f>,
<(ab)d> : <(cf)>, <(_f)cb>
sequential patterns: <(ab)c>, <(ab)d>, <(ab)f>, <(ab)dc>

Similarly, for <ac>, <ad>, <af>

PrefixSpan: Example (Contd.)

Sequence_ID	*Sequence*
1	$\langle a(abc)(ac)d(cf)\rangle$
2	$\langle (ad)c(bc)(ae)\rangle$
3	$\langle (ef)(ab)(df)cb\rangle$
4	$\langle eg(af)cbc\rangle$

1. Find length-1 patterns
<a>: 4, <b>: 4, <c>: 4, <d>: 3, <e>: 3, <f>: 3

2. Partition the search space, based on prefix

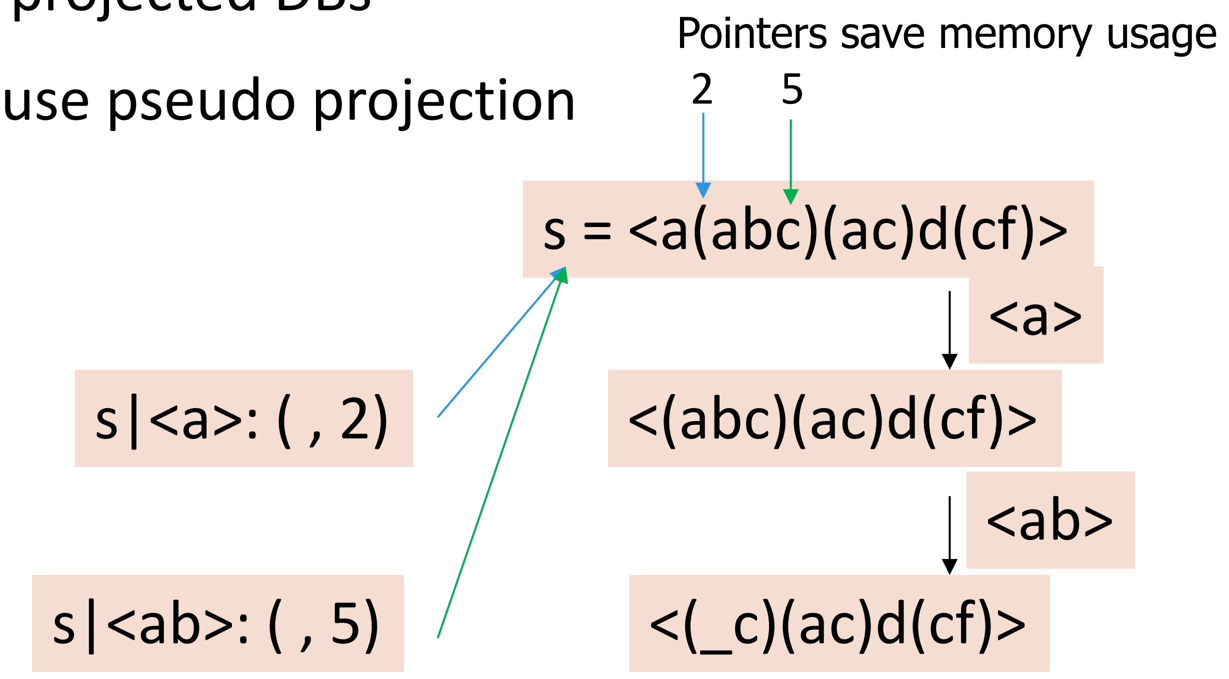
3. Find subsets of patterns: <a>-projected db, frequent <a>: 2, <b>: 4, <_b>: 2, <c>: 4, <d>: 2, <f>: 2

4. All patterns found recursively

prefix	projected database	sequential patterns
$\langle a\rangle$	$\langle (abc)(ac)d(cf)\rangle$, $\langle (\_d)c(bc)(ae)\rangle$, $\langle (\_b)(df)cb\rangle$, $\langle (\_f)cbc\rangle$	$\langle a\rangle$, $\langle aa\rangle$, $\langle ab\rangle$, $\langle a(bc)\rangle$, $\langle a(bc)a\rangle$, $\langle aba\rangle$, $\langle abc\rangle$, $\langle (ab)\rangle$, $\langle (ab)c\rangle$, $\langle (ab)d\rangle$, $\langle (ab)f\rangle$, $\langle (ab)dc\rangle$, $\langle ac\rangle$, $\langle aca\rangle$, $\langle acb\rangle$, $\langle acc\rangle$, $\langle ad\rangle$, $\langle adc\rangle$, $\langle af\rangle$
$\langle b\rangle$	$\langle (\_c)(ac)d(cf)\rangle$, $\langle (\_c)(ae)\rangle$, $\langle (df)cb\rangle$, $\langle c\rangle$	$\langle b\rangle$, $\langle ba\rangle$, $\langle bc\rangle$, $\langle (bc)\rangle$, $\langle (bc)a\rangle$, $\langle bd\rangle$, $\langle bdc\rangle$, $\langle bf\rangle$
$\langle c\rangle$	$\langle (ac)d(cf)\rangle$, $\langle (bc)(ae)\rangle$, $\langle b\rangle$, $\langle bc\rangle$	$\langle c\rangle$, $\langle ca\rangle$, $\langle cb\rangle$, $\langle cc\rangle$
$\langle d\rangle$	$\langle (cf)\rangle$, $\langle c(bc)(ae)\rangle$, $\langle (\_f)cb\rangle$	$\langle d\rangle$, $\langle db\rangle$, $\langle dc\rangle$, $\langle dcb\rangle$
$\langle e\rangle$	$\langle (\_f)(ab)(df)cb\rangle$, $\langle (af)cbc\rangle$	$\langle e\rangle$, $\langle ea\rangle$, $\langle eab\rangle$, $\langle eac\rangle$, $\langle eacb\rangle$, $\langle eb\rangle$, $\langle ebc\rangle$, $\langle ec\rangle$, $\langle ecb\rangle$, $\langle ef\rangle$, $\langle efb\rangle$, $\langle efc\rangle$, $\langle efcb\rangle$.
$\langle f\rangle$	$\langle (ab)(df)cb\rangle$, $\langle cbc\rangle$	$\langle f\rangle$, $\langle fb\rangle$, $\langle fbc\rangle$, $\langle fc\rangle$, $\langle fcb\rangle$

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

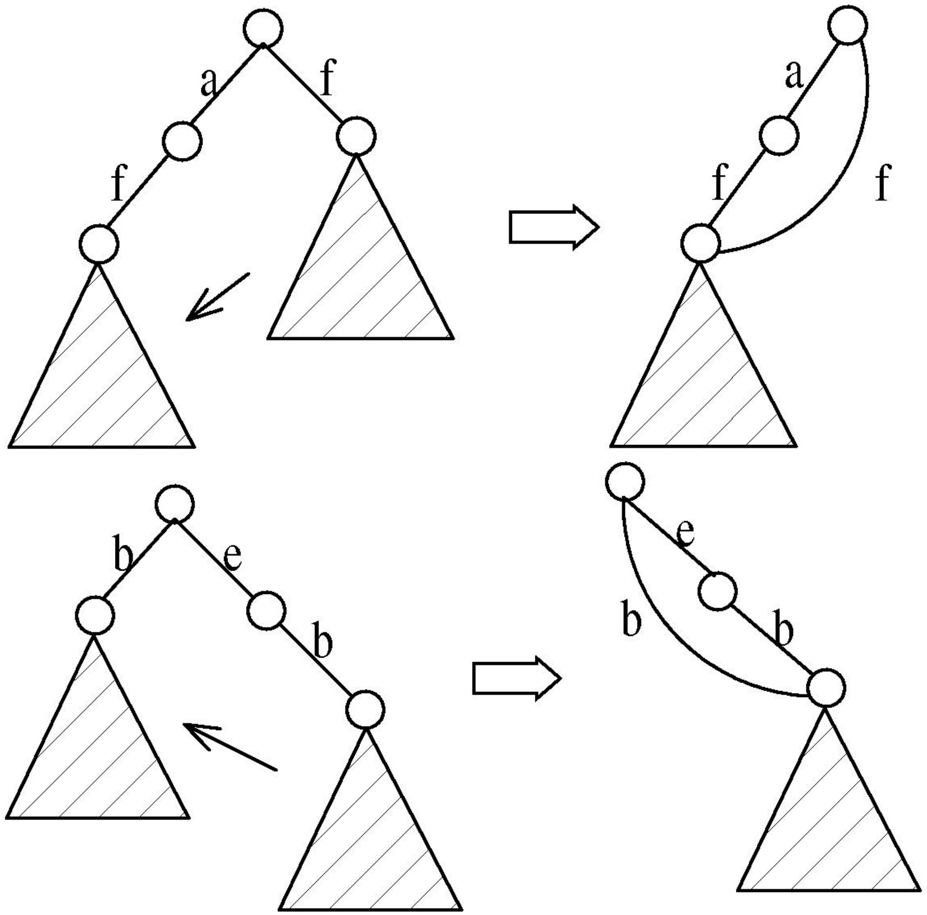


CloSpan: Mining Closed Sequential Patterns

- A closed sequential pattern *s*: There exists no superpattern *s'* such that $s' \supset s$, and *s'* and *s* have the same support
- Which ones are closed? <abc>: 20, <abcd>:20, <abcde>: 15

CloSpan: Mining Closed Sequential Patterns

- Why directly mine closed sequential patterns?
  - Reduce # of (redundant) patterns
  - Attain the same expressive power
- Property P_1: If $s \supset s_1$, 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)

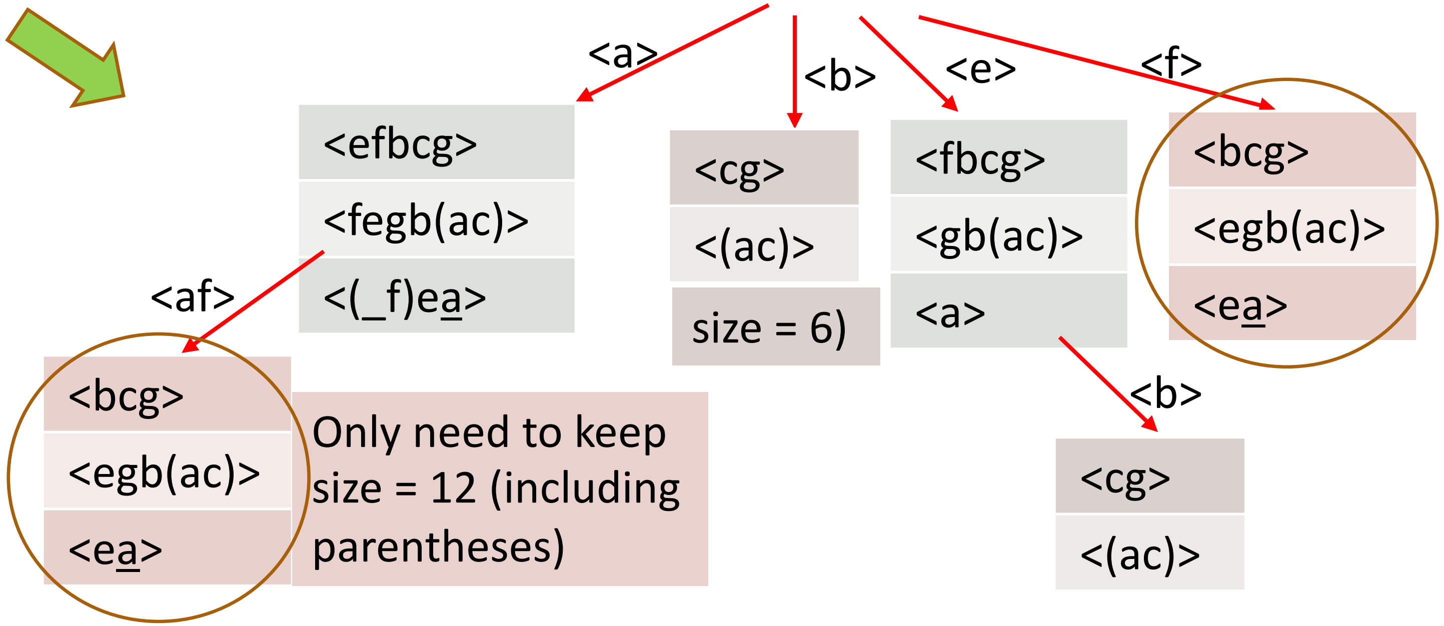


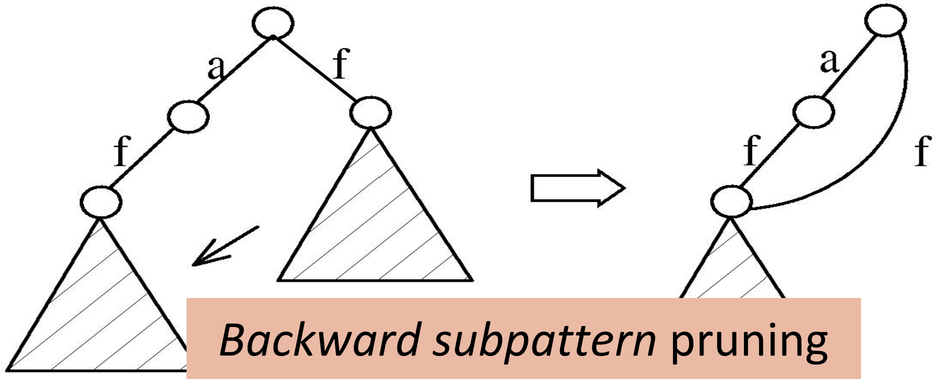
CloSpan: When Two Projected DBs Have the Same Size

- If $s \supset s_1$, s is closed iff two project DBs have the same size
  - When two projected sequence DBs have the same size?
    - Here is one example:

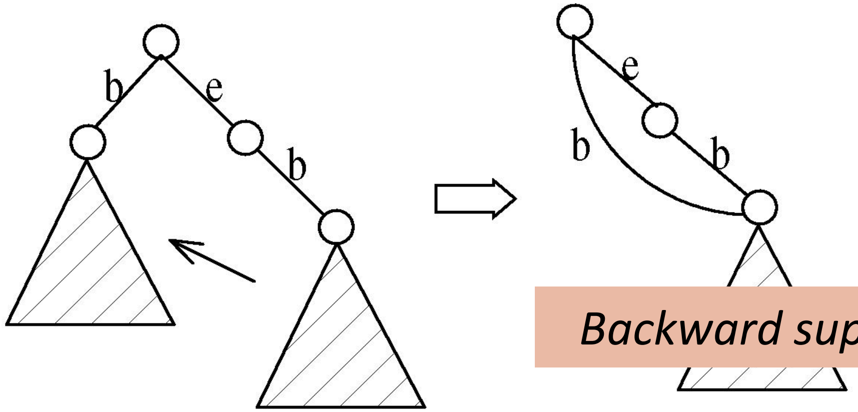
ID	Sequence
1	<aefbcg>
2	<afegb(ac)>
3	<(af)e<u>a</u>>

min_sup = 2





Backward subpattern pruning



Backward superpattern pruning

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) $\geq$ 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

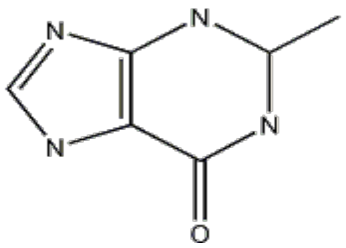
- Episodes and regular expressions: Alternative to seq. patterns
  - Serial episodes: AB ⟵ a total order relationship: first A then B
  - Parallel episodes: A|B ⟵ a partial order relationship: A and B can be in any order
  - Regular expressions: (A|B)C*(DE) ⟵ (DE) means D, E happen in the same time window
- E.g. Given a large shopping sequence database, one may like to find
  - Suppose the pattern order follows the template (A|B)C*(D E), and
  - Sum of the prices of A, B, C*, D, and E is greater than $100, where C* means C appears *-times
  - How to efficiently mine such episode patterns?

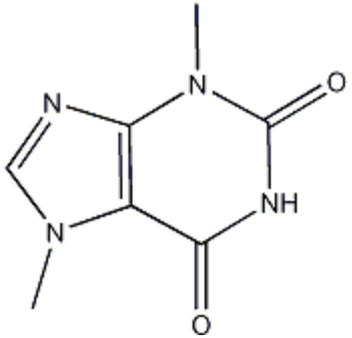
Chapter 6 : Advanced Frequent Pattern Mining

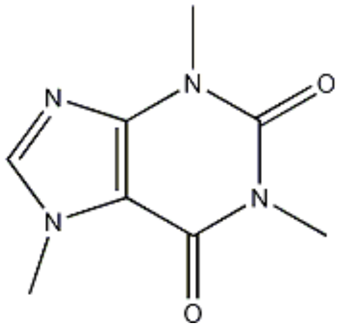
- Mining Diverse Patterns
- Constraint-Based Frequent Pattern Mining
- Sequential Pattern Mining
- Graph Pattern Mining
- Summary

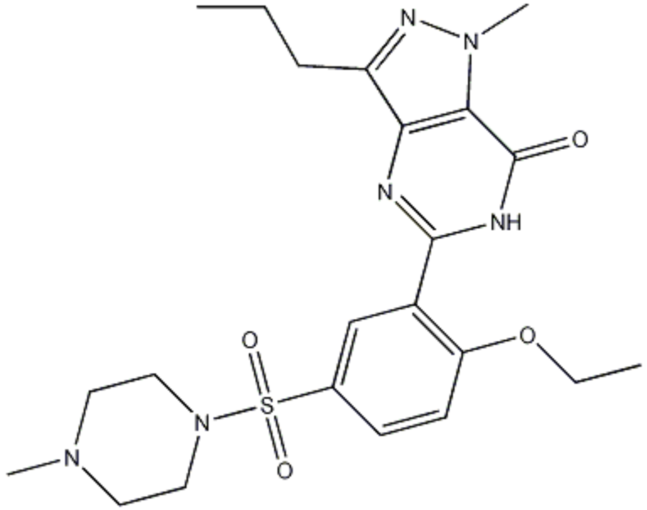
What Is Graph Pattern Mining?

- Chem-informatics:
  - Mining frequent chemical compound structures

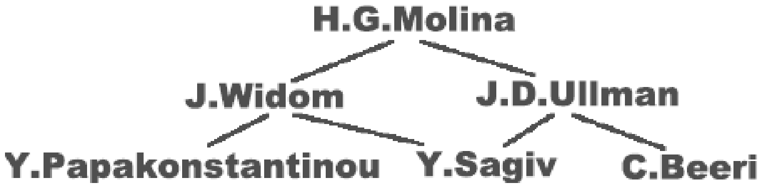


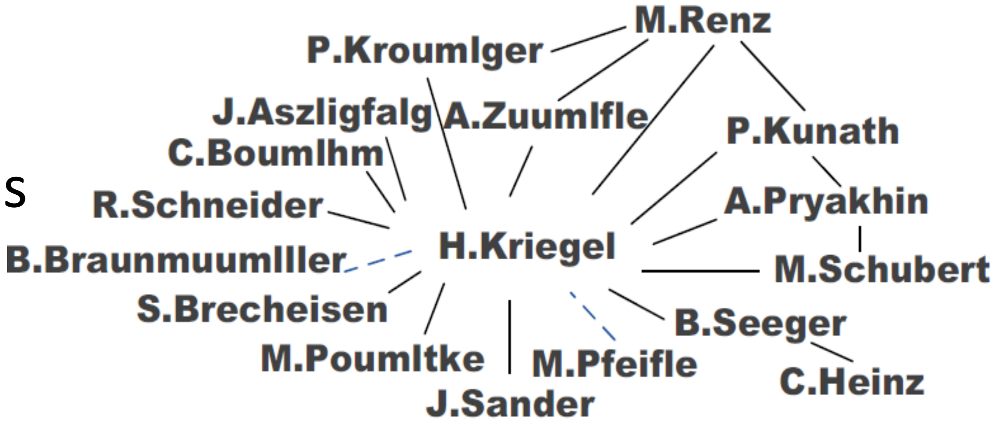






- Social networks, web communities, tweets, …
  - Finding frequent research collaboration subgraphs

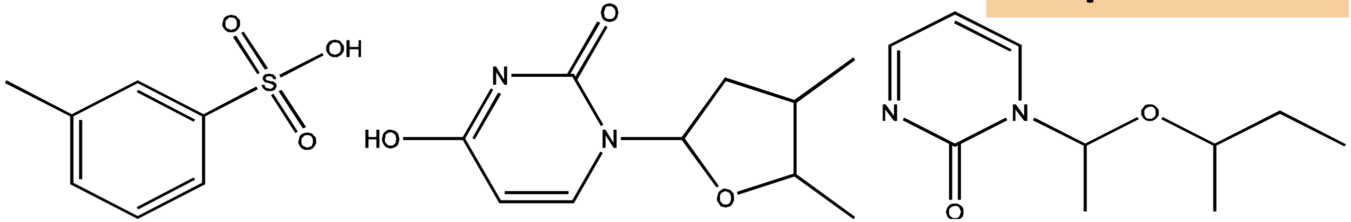




Frequent (Sub)Graph Patterns

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

Graph Dataset

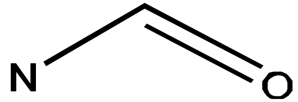


(A) (B) (C)

min_sup = 2

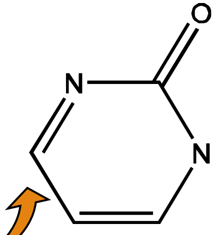
Frequent Graph Patterns

(1)



support = 67%

(2)



Applications of Graph Pattern Mining

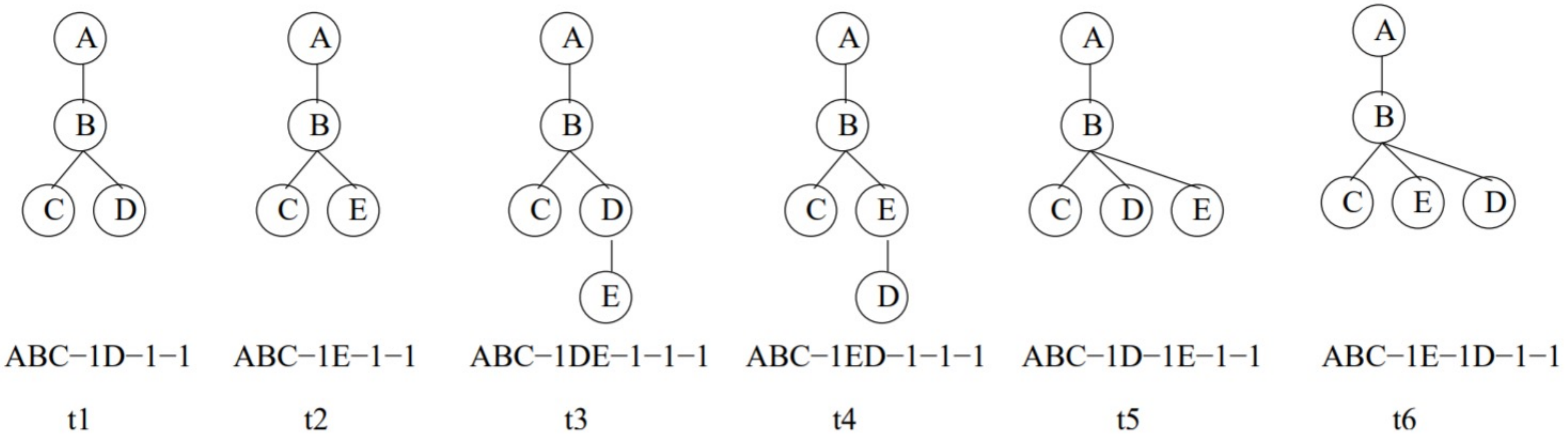
- Bioinformatics
  - Gene networks, protein interactions, metabolic pathways
- Chem-informatics: Mining chemical compound structures
- Social networks, web communities, tweets, …
- Cell phone networks, computer networks, …
- Web graphs, XML structures, Semantic Web, information networks
- Software engineering: 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, 2002])
- Support calculation
  - Store embeddings (e.g., GASTON [Nijssen & Kok, 2004], FFSM [Huan, Wang, & Prins, 2003], MoFa [Borgelt & Berthold, ICDM'02])
- Order of pattern discovery
  - Path → tree → graph (e.g., GASTON [Nijssen & Kok, 2004])

Key Ideas: Frequent Trees

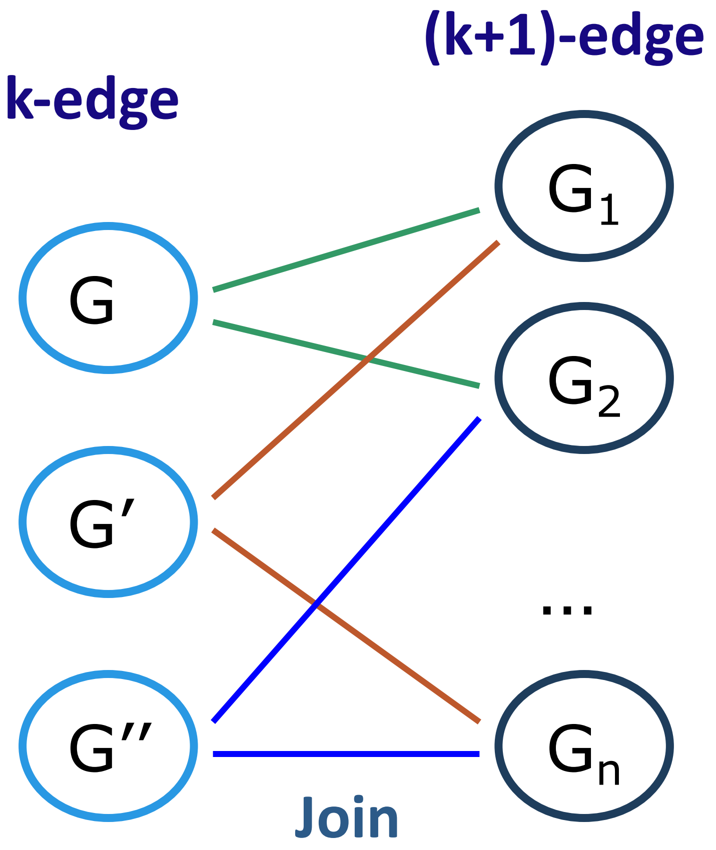
- String encoding (TreeMiner [Zaki, 2002])



- Frequent sequence mining

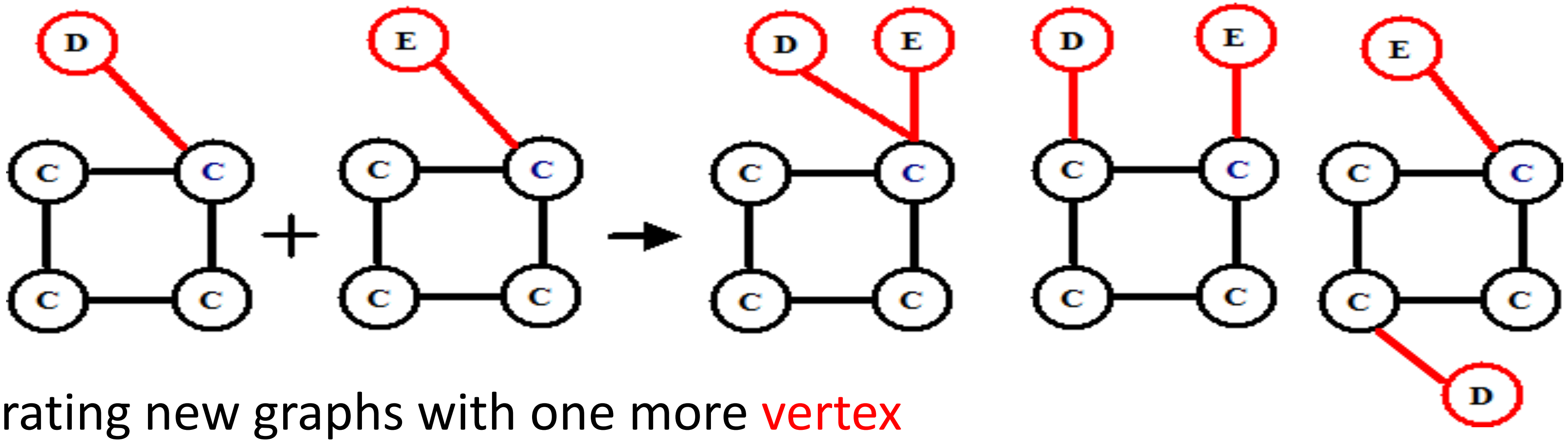
Apriori-Based Approach

- 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

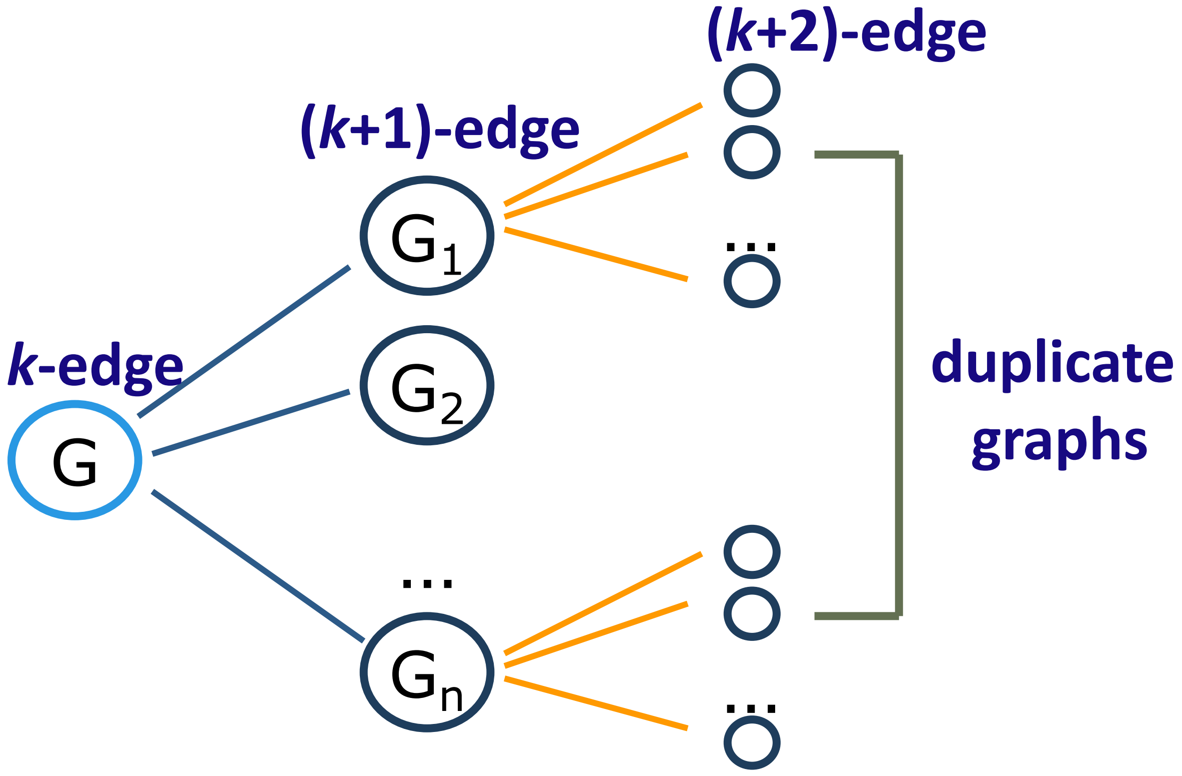
- Methodology: Breadth-search, Apriori joining two size-*k* graphs
  - Many possibilities at generating size-(*k*+1) candidate graphs



- Generating new graphs with one more vertex
  - AGM (Inokuchi, Washio, & Motoda, PKDD'00)
- Generating new graphs with one more edge
  - FSG (Kuramochi & 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)



Pattern Growth on Graphs

- Pattern Growth can use BFS (like Apriori) or DFS
- Pattern growth by adding a new edge
  - May or may not introduce a new vertex
- General strategy: PatternGrowthGraph
  - May not be able to avoid duplicate graphs, can be inefficient

Algorithm: PatternGrowthGraph(g, D, minsup, S)

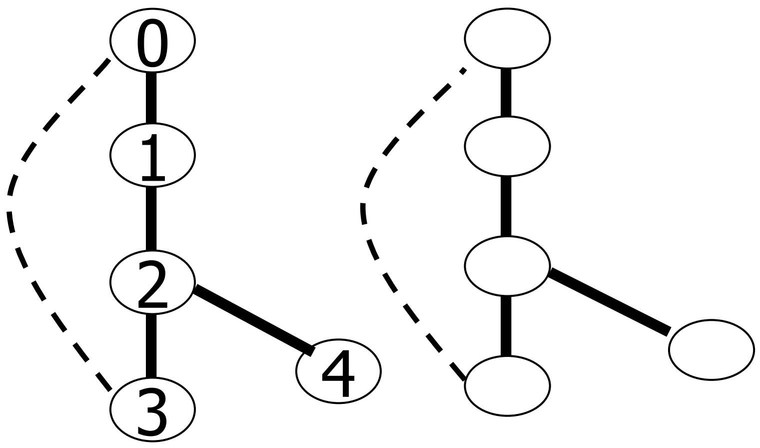
Input: A frequent graph g, a graph data set D, and the support threshold *minsup*.
Output: The frequent graph set S.

if $g \in S$ **then return**;
else insert g to S;
scan D once, find all the edges e such that g can be extended to $g \diamond_x e$;
for each frequent $g \diamond_x e$ **do**
    Call PatternGrowthGraph($g \diamond_x e$, D, minsup, S);
return;

Figure 6.15: PatternGrowthGraph.

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



e_0: (0,1)

e_1: (1,2)

e_2: (2,3)

e_3: (3,0)

e_4: (2,4)

gSpan: Graph Traversal, DFS subscripting

- Traverses the graph by DFS (Depth First Search), starting from some initial vertex
  - Create a full DFS tree
- Different DFS trees for the same graph
- Visit order of vertices in DFS tree noted as v_0 (root), v_1, v_2, …, v_n (right most vertex)
- DFS tree T is the DFS subscripting of G

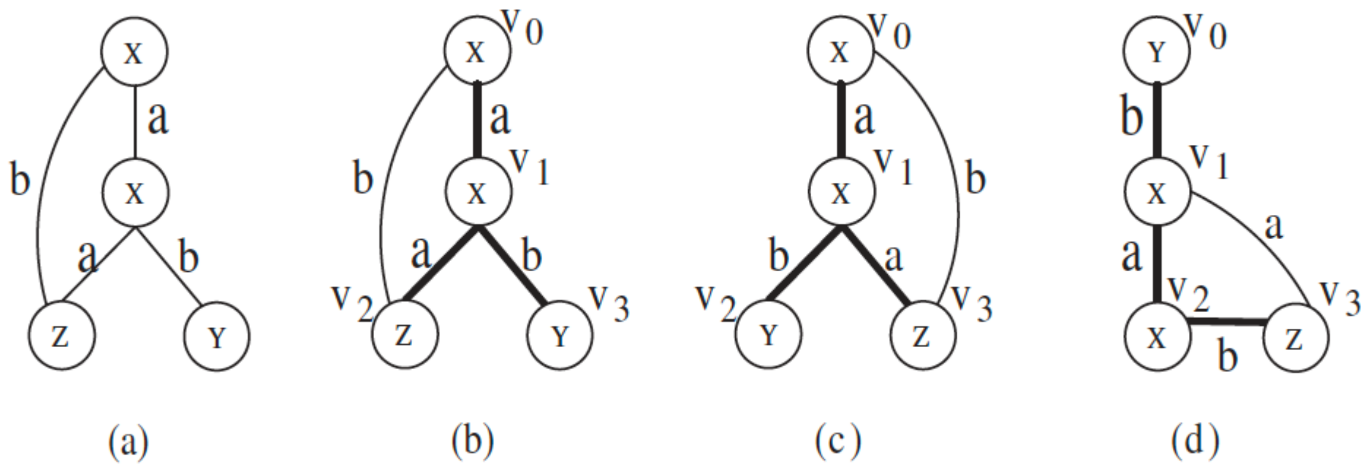


Figure 6.16: DFS subscripting.

Right most path:
Straight path from v_0 to v_n

(b), (c) v_0, v_1, v_3
(d) v_0, v_1, v_2, v_3

Graph Extension: Backward and Forward

- Restrict edge extension, on DFS tree T
- Backward extension: Right-most vertex & vertices in right most edge
- Forward extension: New vertex & vertices in the right most path

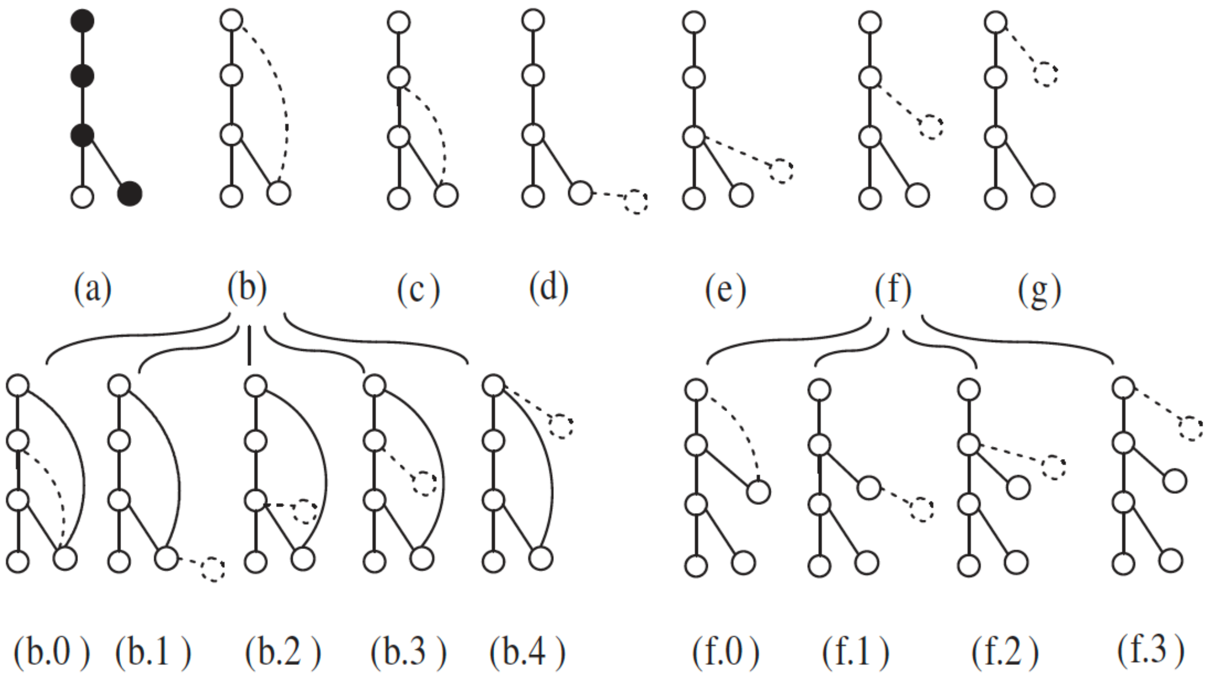


Figure 6.17: Right-Most Extension

Handling Multiple DFS Trees: DFS Codes

- Convert each DFS tree subscripted graph to an edge sequence: DFS code
- Two kinds of orders:
  - Edge order: Forward edges in DFS, add backward edges before forward edges from vertex
  - Sequence order: Order among edge sequences, i.e., graphs
    - Forward edge order for 6.16(b): (0,1), (1,2), (1,3)
    - Edge order for 6.16(b): (0,1), (1,2), (2,0), (1,3)
    - Represent edge as a tuple $(i,j,l_i,l_{(i,j)},l_j)$, order based on
      - Edge order (i,j), labels in vertices l_i, l_j and edges $l_{(i,j)}$

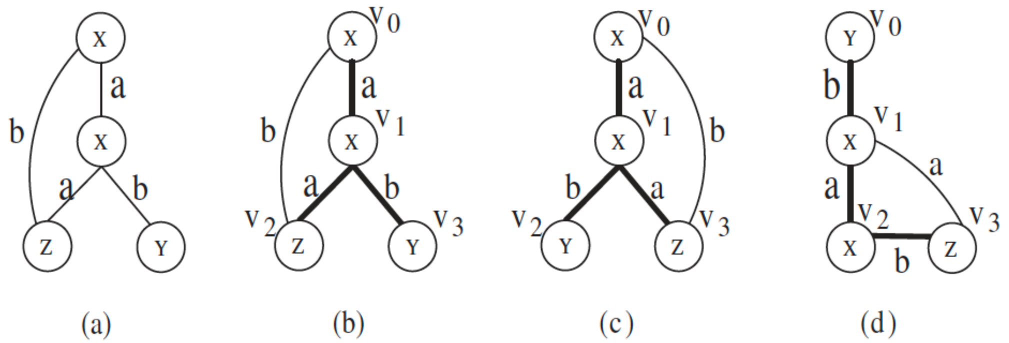


Figure 6.16: DFS subscripting.

edge	γ_0	γ_1	γ_2
e_0	$(0,1,X,a,X)$	$(0,1,X,a,X)$	$(0,1,Y,b,X)$
e_1	$(1,2,X,a,Z)$	$(1,2,X,b,Y)$	$(1,2,X,a,X)$
e_2	$(2,0,Z,b,X)$	$(1,3,X,a,Z)$	$(2,3,X,b,Z)$
e_3	$(1,3,X,b,Y)$	$(3,0,Z,b,X)$	$(3,1,Z,a,X)$

Table 6.6: DFS code for Figure 6.16(b), 6.16(c), and 6.16(d).

DFS Codes: Lexicographic Order

- One graph may have several DFS codes
- Edge as a 5-tuple: $(i,j,l_i,l_{(i,j)},l_j)$, order based on
  - Edge order (i,j), labels in vertices l_i, l_j and edges $l_{(i,j)}$
- Ordering
  - Edge order (i,j)
  - Vertex label l_i
  - Edge label $l_{(i,j)}$,
  - Vertex label l_j
- Use the minimum DFS code (G) for graph G
- Graphs G, G’ are isomorphic if (G) = (G’)

DFS Codes: Lexicographic Order

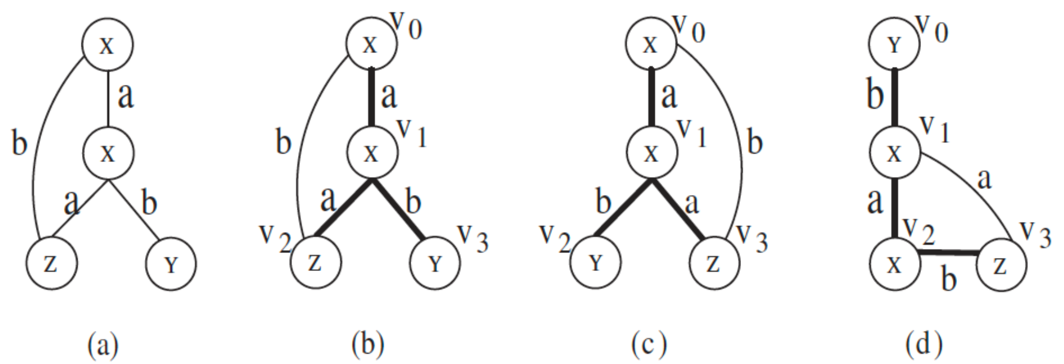


Figure 6.16: DFS subscripting.

edge	γ_0	γ_1	γ_2
e_0	$(0,1,X,a,X)$	$(0,1,X,a,X)$	$(0,1,Y,b,X)$
e_1	$(1,2,X,a,Z)$	$(1,2,X,b,Y)$	$(1,2,X,a,X)$
e_2	$(2,0,Z,b,X)$	$(1,3,X,a,Z)$	$(2,3,X,b,Z)$
e_3	$(1,3,X,b,Y)$	$(3,0,Z,b,X)$	$(3,1,Z,a,X)$

Table 6.6: DFS code for Figure 6.16(b), 6.16(c), and 6.16(d).

- We have: (0,1,X,a,X) < (0,1,Y,b,Y) so that γ_0, $\gamma_1 < \gamma_2$
- We have: (1,2,X,a,Z) < (1,2,X,b,Y) so that $\gamma_0 < \gamma_1$
- Final ordering: $\gamma_0 < \gamma_1 < \gamma_2$

  - Use the minimum DFS code (G) for graph G
  - Graphs G, G’ are isomorphic if (G) = (G’)
  - Right most extensions only on the minimum DFS codes, guarantees completeness

Lexicographic Search Tree

- Arranging DFS codes in a search tree, with right most extensions
- Each node: DFS code encoding a graph
- Each edge: Right-most extension from (k-1)- to k-length DFS code
- Left sibling < right sibling, in lexicographic order
- Work with minimum DFS codes
  - Can drop non-minimum DFS codes
  - Key difference between PatternGrowth and gSpan

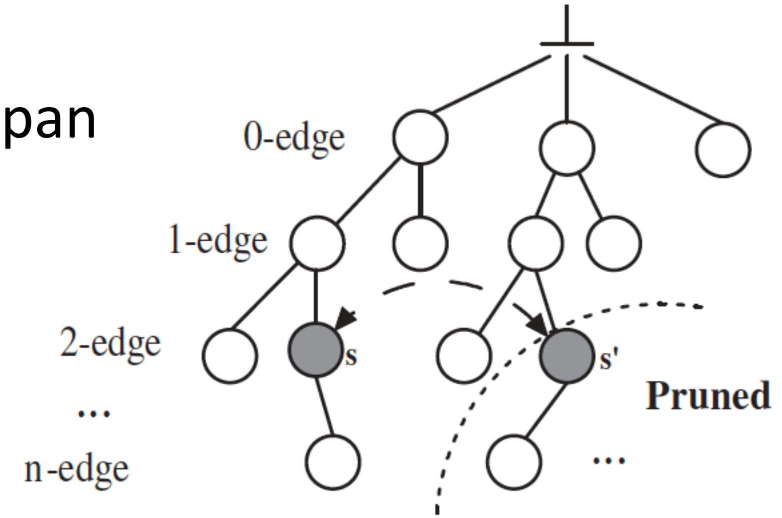


Figure 6.18: Lexicographic search tree.

gSpan Algorithm

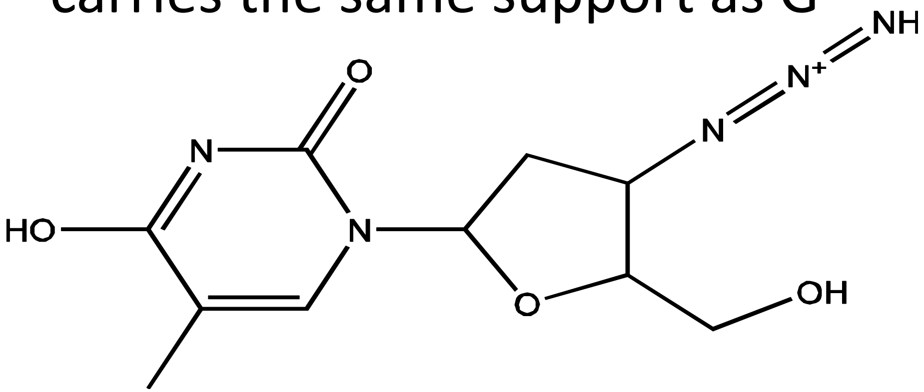
Algorithm: gSpan(s, D, **minsup**, S)

Input: A DFS code s, a graph data set D, and .
Output: The frequent graph set S.

1: **if** $s \neq dfs(s)$, **then**
2: **return**;
3: insert s into S;
4: set C to $\varnothing$;
5: scan D once, find all the edges e such that s can be *right-most* extended to $s \diamond_r e$; insert $s \diamond_r e$ into C and count its frequency;
6: sort C in DFS lexicographic order;
7: **for each** frequent $s \diamond_r e$ in C **do**
8: Call gSpan($s \diamond_r e$, D, minsup, S);
9: **return**;

Why Mine Closed Graph Patterns?

- 2^n subgraphs -> *closed frequent subgraphs*
- A frequent graph G is *closed* if there exists no supergraph of G that carries the same support as G

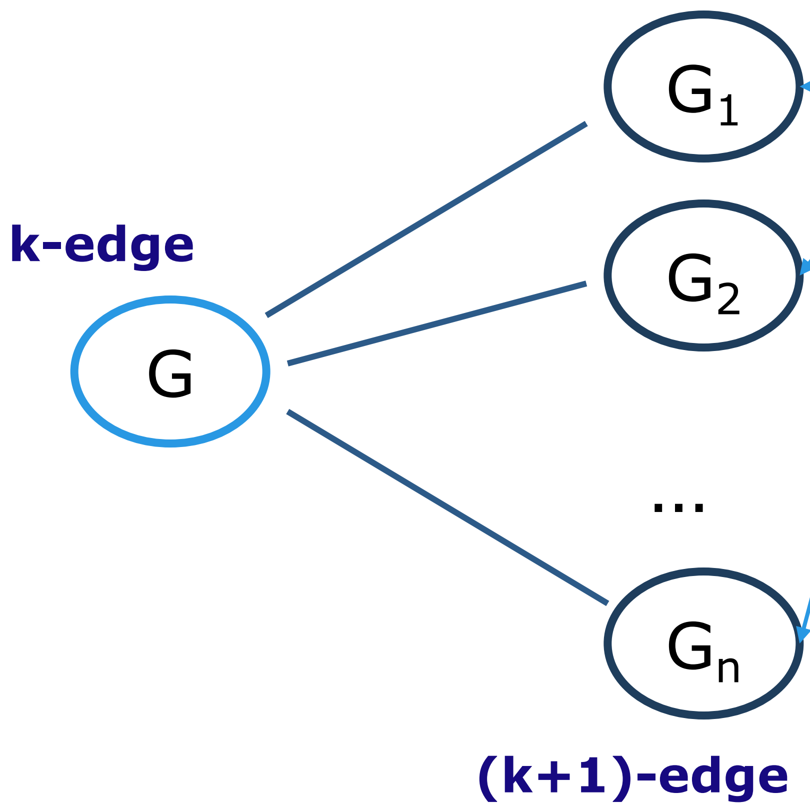


If this subgraph is *closed* in the graph dataset, it implies that none of its frequent super-graphs carries the same support

- *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 (Yan & Han, KDD'03)



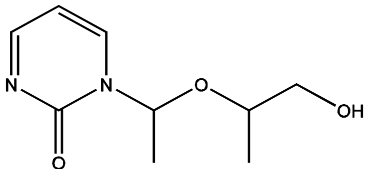
At what condition can we stop searching their children, i.e., early termination?

- 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

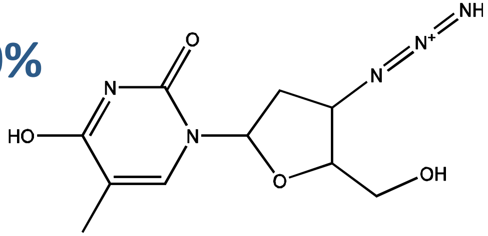
Experiment and Performance Comparison

- 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

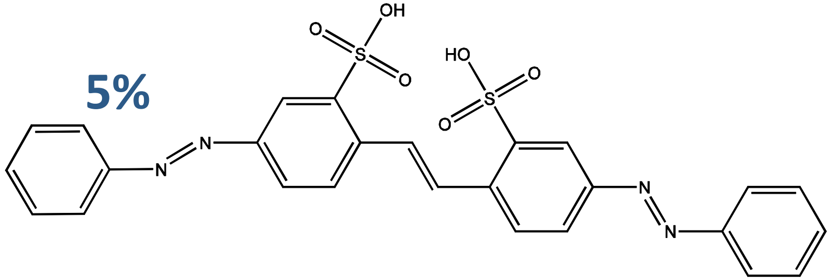
20%



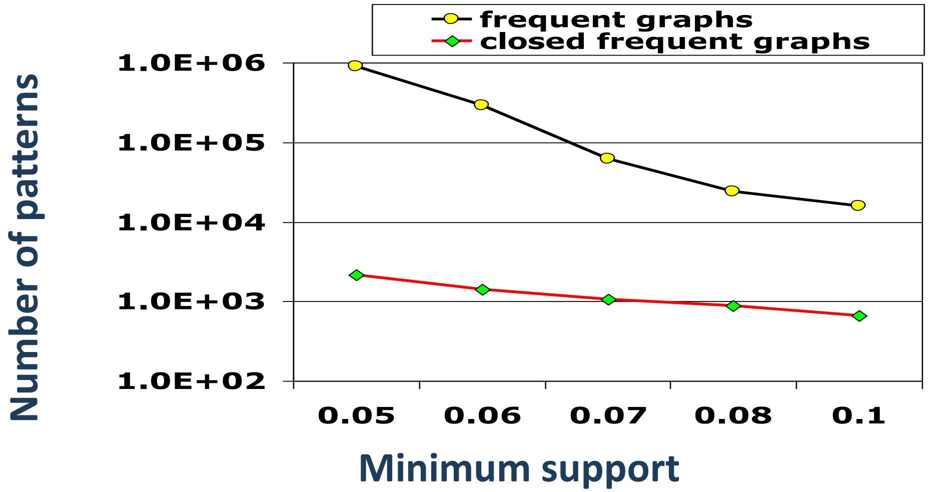
10%



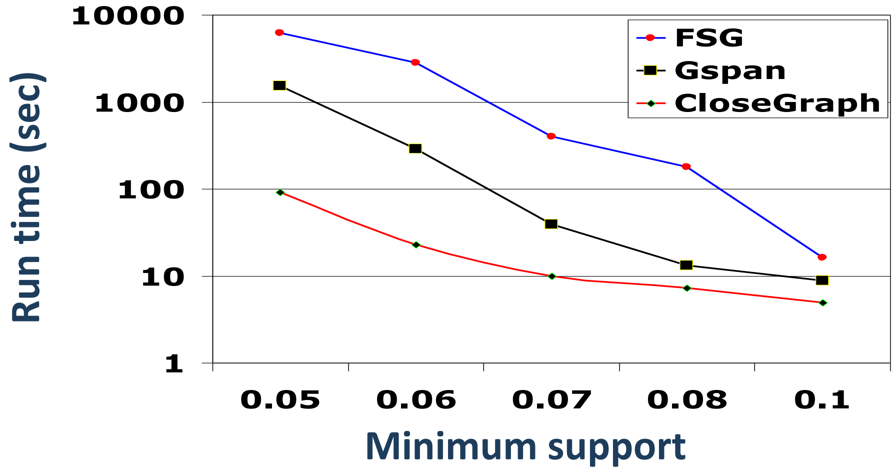
5%



of Patterns: Frequent vs. Closed



Runtime: Frequent vs. Closed



Chapter 6 : Advanced Frequent Pattern Mining

- Mining Diverse Patterns
- Constraint-Based Frequent Pattern Mining
- Sequential Pattern Mining
- Graph Pattern Mining
- Pattern Mining Application: Mining Software Copy-and-Paste Bugs
- Summary

Pattern Mining Application: Software Bug Detection

- **Mining rules from source code**
  - Bugs as deviant behavior (e.g., by statistical analysis)
  - Mining programming rules (e.g., by frequent itemset mining)
  - Mining function precedence protocols (e.g., by frequent subsequence mining)
  - Revealing neglected conditions (e.g., by frequent itemset/subgraph mining)
- **Mining rules from revision histories**
  - By frequent itemset mining
- **Mining copy-paste patterns from source code**
  - Find copy-paste bugs (e.g., CP-Miner [Li et al., OSDI'04]) (to be discussed here)
    - Reference: Z. Li, S. Lu, S. Myagmar, Y. Zhou, "CP-Miner: A Tool for Finding Copy-paste and Related Bugs in Operating System Code", OSDI'04

Application Example: Mining Copy-and-Paste Bugs

- Copy-pasting is common
  - 12% in Linux file system
  - 19% in X Window system
- Copy-pasted code is error-prone
- Mine *"forget-to-change"* bugs by sequential pattern mining
  - Build a sequence database from source code
  - Mining sequential patterns
  - Finding mismatched identifier names & bugs

```
void __init prom_meminit(void)
{
   ......
   for (i=0; i<n; i++) {
      total[i].adr = list[i].addr;
      total[i].bytes = list[i].size;
      total[i].more = &total[i+1];
   }
   ......
```

```
for (i=0; i<n; i++) {
      taken[i].adr = list[i].addr;
      taken[i].bytes = list[i].size;
      taken[i].more = &total[i+1];
   }
```

Code copy-and-pasted but **forget to change** "id"!

(Simplified example from *linux-2.6.6/arch/sparc/prom/memory.c*)

Courtesy of Yuanyuan Zhou@UCSD

Chapter 6 : Advanced Frequent Pattern Mining

- Mining Diverse Patterns
- Constraint-Based Frequent Pattern Mining
- Sequential Pattern Mining
- Graph Pattern Mining
- Pattern Mining Application: Mining Software Copy-and-Paste Bugs
- Summary

Summary: Advanced Frequent Pattern Mining

- Mining Diverse Patterns
  - Mining Multiple-Level Associations
  - Mining Multi-Dimensional Associations
  - Mining Quantitative Associations
  - Mining Negative Correlations
  - Mining Compressed and Redundancy-Aware Patterns
- Sequential Pattern Mining
  - Sequential Pattern and Sequential Pattern Mining
  - GSP: Apriori-Based Sequential Pattern Mining
  - SPADE: Sequential Pattern Mining in Vertical Data Format
  - PrefixSpan: Sequential Pattern Mining by Pattern-Growth
  - CloSpan: Mining Closed Sequential Patterns
- Constraint-Based Frequent Pattern Mining
  - Why Constraint-Based Mining?
  - Constrained Mining with Pattern Anti-Monotonicity
  - Constrained Mining with Pattern Monotonicity
  - Constrained Mining with Data Anti-Monotonicity
  - Constrained Mining with Succinct Constraints
  - Constrained Mining with Convertible Constraints
  - Handling Multiple Constraints
  - Constraint-Based Sequential-Pattern Mining
- Graph Pattern Mining
  - Graph Pattern and Graph Pattern Mining
  - Apriori-Based Graph Pattern Mining Methods
  - gSpan: A Pattern-Growth-Based Method
  - CloseGraph: Mining Closed Graph Patterns
- Pattern Mining Application: Mining Software Copy-and-Paste Bugs

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