CSE 5243 INTRO. TO DATA MINING

Advanced Pattern Mining

(Chapter 7)

Yu Su, CSE@The Ohio State University

Slides adapted from UIUC CS412 by Prof. Jiawei Han and OSU CSE5243 by Prof. Huan Sun

Chapter 7 : 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

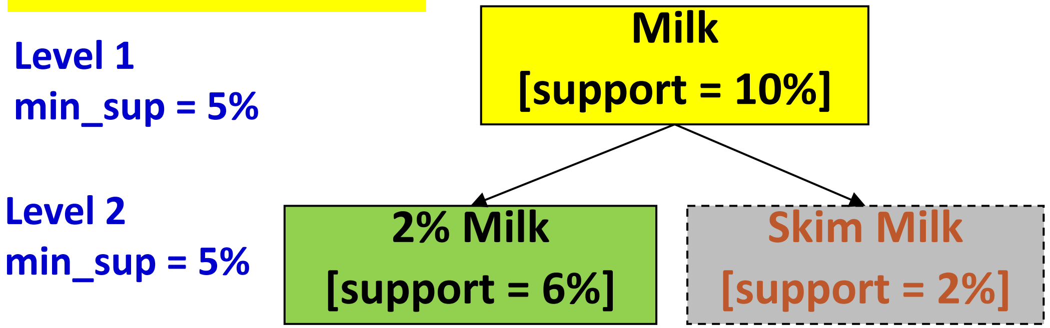
Mining Diverse Patterns

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

Mining Multiple-Level Frequent Patterns

- Items often form hierarchies
  - Ex.: Dairyland 2% milk; Wonder wheat bread
- How to set min-support thresholds?

Uniform support



- Uniform min-support across multiple levels (reasonable?)

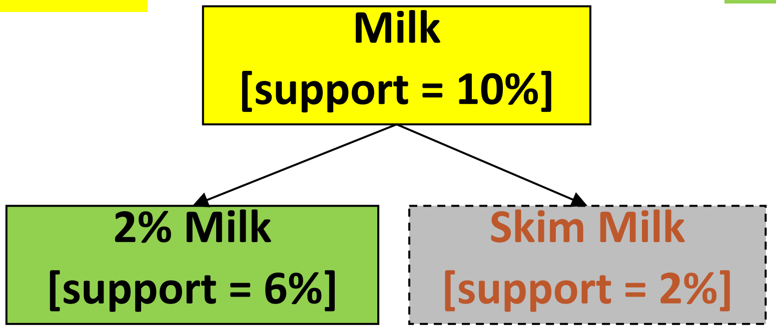
Mining Multiple-Level Frequent Patterns

- Items often form hierarchies
  - Ex.: Dairyland 2% milk; Wonder wheat bread
- How to set min-support thresholds?

Uniform support

Level 1
min_sup = 5%

Level 2
min_sup = 5%



Reduced support

Level 1
min_sup = 5%

Level 2
min_sup = 1%

- Uniform min-support across multiple levels (reasonable?)
- Level-reduced min-support: Items at the lower level are expected to have lower support

ML/MD Associations with Flexible Support Constraints

- Why flexible support constraints?
  - Real life occurrence frequencies vary greatly
    - Diamond, watch, pens in a shopping basket
  - Uniform support may not be an interesting model
- A flexible model
  - The lower-level, the more dimension combination, and the longer pattern length, usually the smaller support
  - General rules should be easy to specify and understand
  - Special items and special group of items may be specified individually and have higher priority

Multi-level Association: Redundancy Filtering

- Some rules may be redundant due to "ancestor" relationships between items.
- Example
  - milk $\Rightarrow$ wheat bread [support = 8%, confidence = 70%]
  - 2% milk $\Rightarrow$ wheat bread [support = 2%, confidence = 72%]
  - Given the 2% milk sold is about ¼ of milk sold
- We say the first rule is an ancestor of the second rule.
- A rule is redundant if its support and confidence are close to the "expected" value, based on the rule's ancestor.

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”)

Mining 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

Mining 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 Negatively Correlated Patterns

- A (relative) support-based definition
  - If itemsets A and B are both frequent but rarely occur together, i.e., $sup(A \cup B) << sup(A) \times sup(B)$
  - Then A and B are negatively correlated

Defining Negative Correlated Patterns

- A (relative) support-based definition
  - If itemsets A and B are both frequent but rarely occur together, i.e., $sup(A \cup B) \ll sup(A) \times sup(B)$
  - Then A and B are negatively correlated

Does this remind you the definition of *lift*?

- Is this a good definition for large transaction datasets?

Defining Negative Correlated Patterns

- A (relative) 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)

Defining Negative Correlated Patterns

- A (relative) 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
    - $s(A \cup B) = 0.005$, $s(A) \times s(B) = 0.25$, $s(A \cup B) << s(A) \times s(B)$
  - But when there are 10^5 transactions, we have
    - $s(A \cup B) = 1/10^5$, $s(A) \times s(B) = 1/10^3 \times 1/10^3$, $s(A \cup B) > s(A) \times s(B)$
  - 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

Which measure should we use?

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

Definition 7.3: Suppose that itemsets X and Y are both frequent, that is, $sup(X) \geq min\_sup$ and $sup(Y) \geq min\_sup$, where min_sup is the minimum support threshold. If $(P(X|Y) + P(Y|X))/2 < \epsilon$, where ϵ is a negative pattern threshold, then pattern $X \cup Y$ is a **negatively correlated pattern.** □

Chapter 7 : 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

Constraint-based Data Mining

- Finding all the patterns in a database autonomously? — unrealistic!
  - The patterns could be too many but not focused!

Constraint-based Data Mining

- Finding all the patterns in a database autonomously? — unrealistic!
  - The patterns could be too many but not focused!

- Data mining should be an interactive process
  - User directs what to be mined using a data mining query language (or a graphical user interface)

Constraint-based Data Mining

- Finding all the patterns in a database autonomously? — unrealistic!
  - The patterns could be too many but not focused!
- Data mining should be an interactive process
  - User directs what to be mined using a data mining query language (or a graphical user interface)
- Constraint-based mining
  - User flexibility: provides constraints on what to be mined
  - System optimization: explores such constraints for efficient mining—constraint-based mining

Categories of Constraints

CONSTRAINT 1 (ITEM CONSTRAINT). *An* item constraint *specifies what are the particular individual or groups of items that should or should not be present in the pattern.* □

For example, a dairy company may be interested in patterns containing only dairy products, when it mines transactions in a grocery store.

CONSTRAINT 2 (LENGTH CONSTRAINT). *A* length constraint *specifies the requirement on the length of the patterns, i.e., the number of items in the patterns.* □

For example, when mining classification rules for documents, a user may be interested in only frequent patterns with at least 5 keywords, a typical length constraint.

Categories of Constraints

CONSTRAINT 3 (MODEL-BASED CONSTRAINT). *A* model-based constraint *looks for patterns which are sub- or super-patterns of some given patterns (models).* □

For example, a travel agent may be interested in what other cities that a visitor is likely to travel if s/he visits both Washington and New York city. That is, they want to find frequent patterns which are super-patterns of {Washington, New York city}.

CONSTRAINT 4 (AGGREGATE CONSTRAINT). *An* aggregate constraint *is on an aggregate of items in a pattern, where the aggregate function can be* SUM, AVG, MAX, MIN, *etc.* □

For example, a marketing analyst may like to find frequent patterns where the average price of all items in each pattern is over $100.

Constrained Frequent Pattern Mining: A Mining Query Optimization Problem

- Given a frequent pattern mining query with a set of constraints C, the algorithm should be
  - sound: it only finds frequent sets that satisfy the given constraints C
  - complete: all frequent sets satisfying the given constraints C are found

Constrained Frequent Pattern Mining: A Mining Query Optimization Problem

- Given a frequent pattern mining query with a set of constraints C, the algorithm should be
  - sound: it only finds frequent sets that satisfy the given constraints C
  - complete: all frequent sets satisfying the given constraints C are found

- A naïve solution
  - First find all frequent sets, and then test them for constraint satisfaction

The Apriori Algorithm — Example

Database D

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

Scan D

C_1

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

L_1

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

C_2

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

Scan D

C_2

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

L_2

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

C_3

itemset
{2 3 5}

Scan D

L_3

itemset	sup
{2 3 5}	2

Naïve Algorithm: Apriori + Constraint (Naïve Solution)

Database D

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

Scan D →

C_1

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

→

L_1

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

C_2

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

← Scan D

C_2

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

←

L_2

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

C_3

itemset
{2 3 5}

Scan D →

L_3

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

Constraint:

Sum(S.price) < 5

Constrained Frequent Pattern Mining: A Mining Query Optimization Problem

- Given a frequent pattern mining query with a set of constraints C, the algorithm should be
  - sound: it only finds frequent sets that satisfy the given constraints C
  - complete: all frequent sets satisfying the given constraints C are found
- A naïve solution
  - First find all frequent sets, and then test them for constraint satisfaction
- More efficient approaches:
  - Analyze the properties of constraints comprehensively
  - Push them as deeply as possible inside the frequent pattern computation.

Anti-Monotonicity in Constraint-Based Mining

- Anti-monotonicity
  - *When an itemset S **violates** the constraint, so does any of its superset*
  - $sum(S.Price) \leq v$ is anti-monotonic?
  - $sum(S.Price) \geq v$ is anti-monotonic?

Anti-Monotonicity in Constraint-Based Mining

- Anti-monotonicity
  - *When an itemset S* ***violates*** *the constraint, so does any of its superset*
  - $sum(S.Price) \leq v$ is anti-monotonic
  - $sum(S.Price) \geq v$ is not anti-monotonic

Anti-Monotonicity in Constraint-Based Mining

- Anti-monotonicity
  - *When an itemset S **violates** the constraint, so does any of its superset*
  - *sum(S.Price)* $\leq$ *v* is anti-monotonic
  - *sum(S.Price)* $\geq$ *v* is not anti-monotonic
- Example. C: range(S.profit) $\leq$ 15 is anti-monotonic
  - *Define range(S.profit) = max(S.profit) – min(S.profit)*
  - Itemset *ab* violates C
  - So does every superset of *ab*

TDB (min_sup=2)

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

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

Which Constraints Are Anti-Monotonic?

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

Practice offline

Monotonicity in Constraint-Based Mining

- Monotonicity
  - *When an intemset S* ***satisfies*** *the constraint, so does any of its superset*
  - *sum(S.Price)* $\geq$ *v* is ?
  - *min(S.Price)* $\leq$ *v* is ?

Monotonicity in Constraint-Based Mining

- Monotonicity
  - *When an intemset S* ***satisfies*** *the constraint, so does any of its superset*
  - $sum(S.Price) \geq v$ is monotonic
  - $min(S.Price) \leq v$ is monotonic

Monotonicity in Constraint-Based Mining

- Monotonicity
  - *When an intemset S **satisfies** the constraint, so does any of its superset*
  - *sum(S.Price)* $\geq v$ is monotonic
  - *min(S.Price)* $\leq v$ is monotonic
- Example. C: range(S.profit) $\geq$ 15
  - Itemset *ab* satisfies C
  - So does every superset of *ab*

TDB (min_sup=2)

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

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

Which Constraints Are Monotonic?

Constraint	Monotonic
$v \in S$	yes
$S \supseteq V$	yes
$S \subseteq V$	no
$\min(S) \leq v$	yes
$\min(S) \geq v$	no
$\max(S) \leq v$	no
$\max(S) \geq v$	yes
$\text{count}(S) \leq v$	no
$\text{count}(S) \geq v$	yes
$\text{sum}(S) \leq v$ ($a \in S, a \geq 0$)	no
$\text{sum}(S) \geq v$ ($a \in S, a \geq 0$)	yes
$\text{range}(S) \leq v$	no
$\text{range}(S) \geq v$	yes
$\text{avg}(S)\ \theta\ v, \theta \in \{=, \leq, \geq\}$	convertible
$\text{support}(S) \geq \xi$	no
$\text{support}(S) \leq \xi$	yes

Practice offline

The Apriori Algorithm — Example

Database D

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

Scan D →

C_1

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

→

L_1

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

C_2

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

← Scan D

C_2

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

←

L_2

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

C_3

itemset
{2 3 5}

Scan D →

L_3

itemset	sup
{2 3 5}	2

Naïve Algorithm: Apriori + Constraint

Database D

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

Scan D → C_1

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

→ L_1

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

C_2

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

Scan D → C_2

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

→ L_2

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

C_3

itemset
{2 3 5}

Scan D → L_3

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

Constraint:

Sum(S.price) < 5

Pushing the constraint deep into the process

Database D

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

Scan D

C_1

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

L_1

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

C_2

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

Scan D

C_2

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

L_2

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

Why?

C_3

itemset
~~{2 3 5}~~

Scan D

L_3

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

Constraint:

Sum(S.price) < 5

Converting "Tough" Constraints

- Convert tough constraints into anti-monotonic or monotonic ones by properly ordering items

Converting "Tough" Constraints

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

Converting "Tough" Constraints

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

TDB (min_sup=2)

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

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

Convertible Constraints

- Let R be an order of items

- Convertible anti-monotonic
  - If an itemset *S* violates a constraint C, so does every itemset having *S* as a prefix w.r.t. R
  - Ex. $avg(S) \geq v$ w.r.t. item value descending order

Convertible Constraints

- Let R be an order of items

- Convertible anti-monotonic
  - If an itemset *S* violates a constraint C, so does every itemset having *S* as a prefix w.r.t. R
  - Ex. $avg(S) \geq v$ w.r.t. item value descending order

- Convertible monotonic
  - If an itemset *S* satisfies constraint C, so does every itemset having *S* as a prefix w.r.t. R
  - Ex. avg(S) $\leq$ v w.r.t. item value descending order

Strongly Convertible Constraints

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

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

What Constraints Are Convertible?

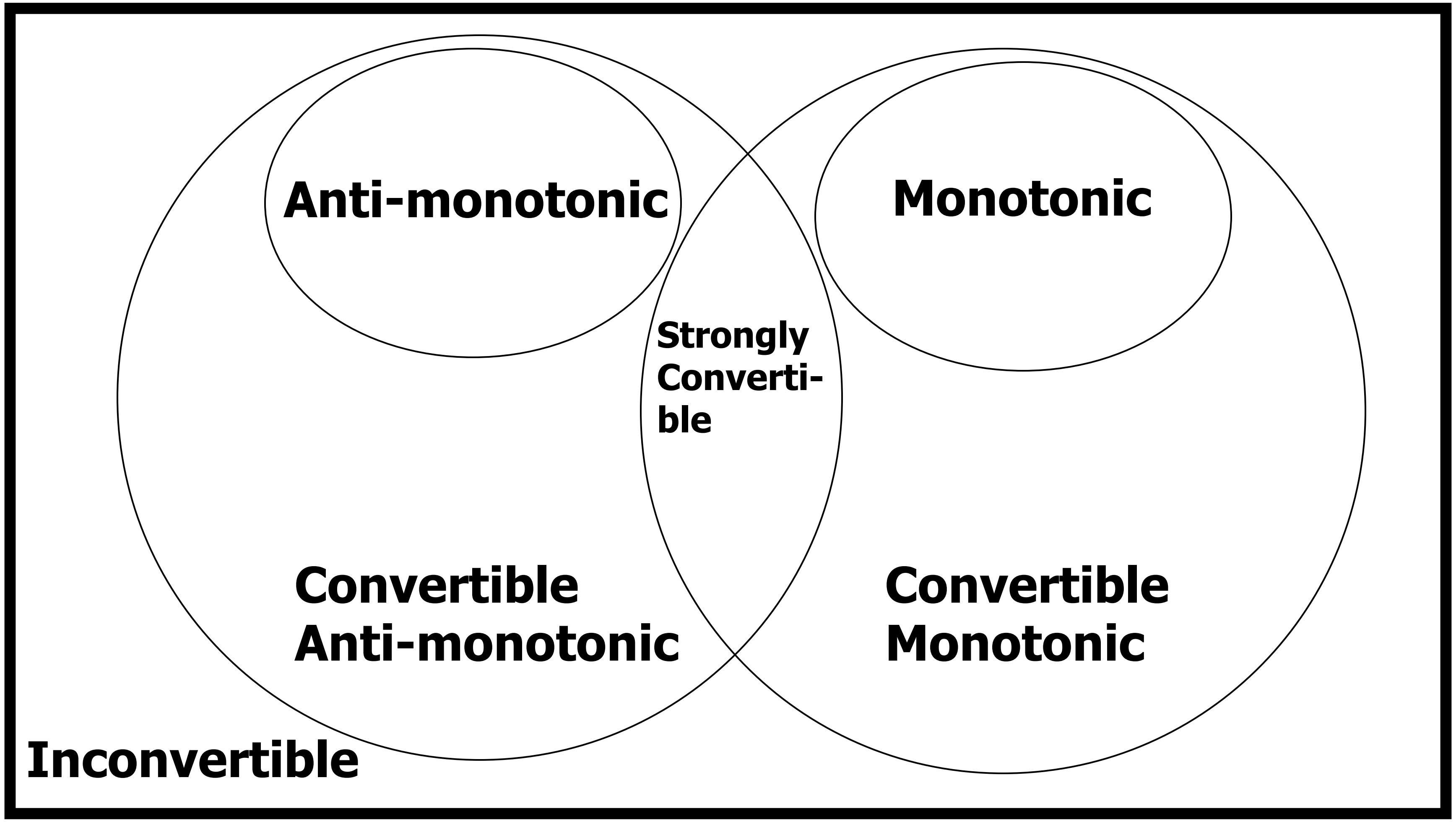
Why?

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

Combining Them Together—A General Picture

Constraint	Antimonotonic	Monotonic
$v \in S$	no	yes
$S \supseteq V$	no	yes
$S \subseteq V$	yes	no
$\min(S) \leq v$	no	yes
$\min(S) \geq v$	yes	no
$\max(S) \leq v$	yes	no
$\max(S) \geq v$	no	yes
$count(S) \leq v$	yes	no
$count(S) \geq v$	no	yes
$sum(S) \leq v$ ($a \in S, a \geq 0$)	yes	no
$sum(S) \geq v$ ($a \in S, a \geq 0$)	no	yes
$range(S) \leq v$	yes	no
$range(S) \geq v$	no	yes
$avg(S)\ \theta\ v, \theta \in \{=, \leq, \geq\}$	convertible	convertible
$support(S) \geq \xi$	yes	no
$support(S) \leq \xi$	no	yes

Classification of Constraints



Mining With Convertible Constraints

- C: avg(S.profit) ≥ 25

TDB (min_sup=2)

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

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

- Scan transaction DB once
  - remove infrequent 1-itemsets
    - Item *h* in transaction 40 is dropped
  - Itemsets *a* and *f* are good

Can Apriori Handle Convertible Constraint?

- A convertible, not monotonic nor anti-monotonic constraint cannot be pushed deep into the an Apriori mining algorithm
  - Within the level wise framework, no direct pruning based on the constraint can be made
  - Itemset {d} violates constraint C: avg(X)>=25
  - **Can we just prune {d} and not consider it afterwards?**

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

Can Apriori Handle Convertible Constraint?

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

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

Mining With Convertible Constraints in FP-Growth Framework

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

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

TDB (min_sup=2)

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

Mining With Convertible Constraints in FP-Growth Framework

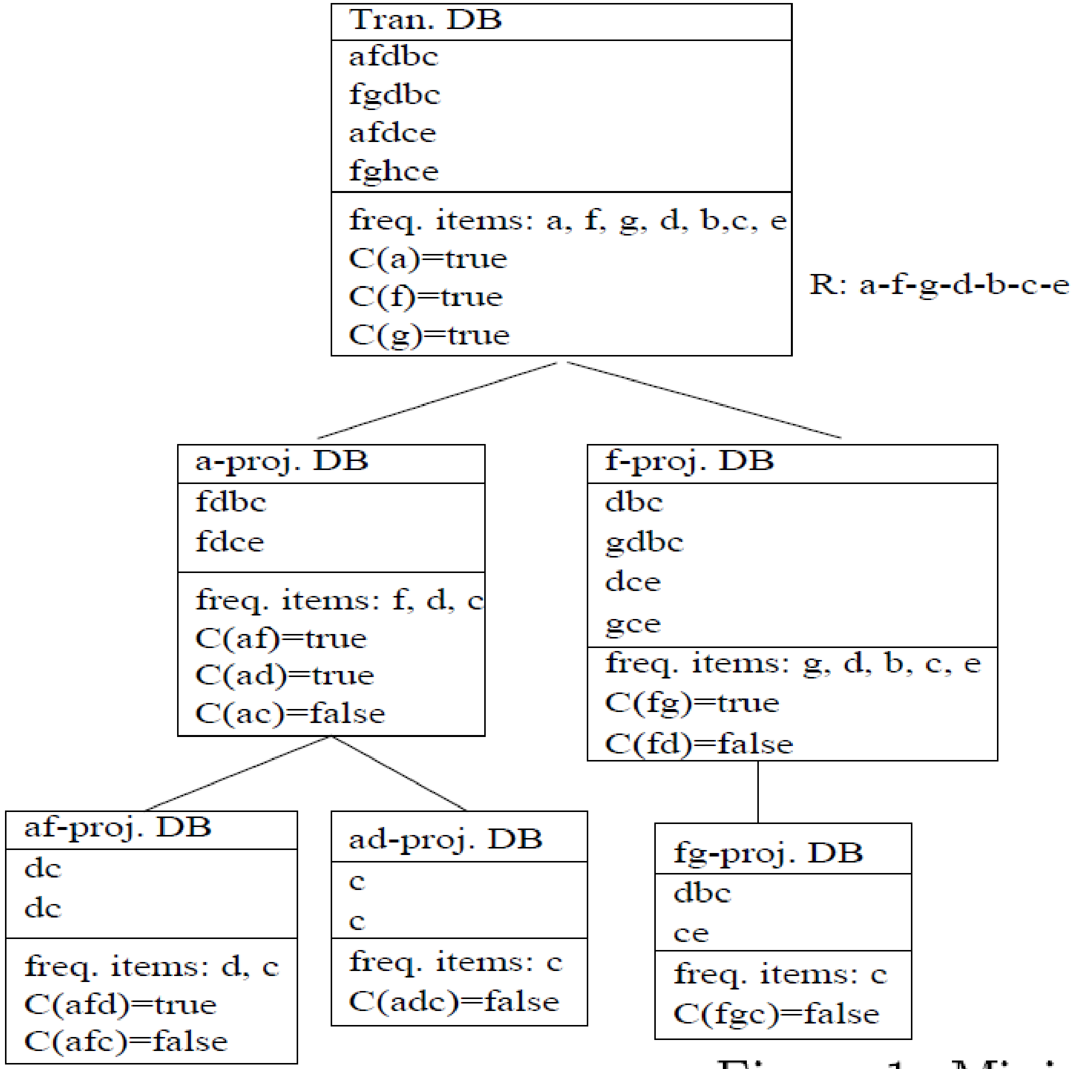


Figure 1: Mining frequent itemsets satisfying constraint $avg(S) \geq 25$.

Growing patterns in R order

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

Constrained Frequent Pattern Mining: A Pattern-Growth View

Jian Pei, Jiawei Han, SIGKDD 2002

Handling Multiple Constraints

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

Chapter 7 : 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

Sequence Databases & 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

Sequence Mining: Description

- Input
  - A database ***D*** of sequences called *data-sequences,* in which:
    - $I=\{i_1, i_2,\ldots,i_n\}$ is the set of items
    - each sequence is a list of transactions ordered by transaction-time
    - each transaction consists of fields: sequence-id, transaction-id, transaction-time and a set of items.

Database $\mathcal{D}$

Sequence-Id	Transaction Time	Items
C1	1	Ringworld
C1	2	Foundation
C1	15	Ringworld Engineers, Second Foundation
C2	1	Foundation, Ringworld
C2	20	Foundation and Empire
C2	50	Ringworld Engineers

Sequence Mining: Description

- Input
  - A database ***D*** of sequences called *data-sequences,* in which:
    - $I=\{i_1, i_2,\ldots,i_n\}$ is the set of items
    - each sequence is a list of transactions ordered by transaction-time
    - each transaction consists of fields: sequence-id, transaction-id, transaction-time and a set of items.
- Problem
  - To discover all the sequential patterns with a user-specified minimum support

Input Database: example

Problem

To discover all the sequential patterns with a user-specified minimum support

Database $\mathcal{D}$

Sequence-Id	Transaction Time	Items
C1	1	Ringworld
C1	2	Foundation
C1	15	Ringworld Engineers, Second Foundation
C2	1	Foundation, Ringworld
C2	20	Foundation and Empire
C2	50	Ringworld Engineers

45% of customers who bought ***Foundation*** will buy ***Foundation and Empire*** within the next month.

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)

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>

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

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

1. An item can occur once at most in an event, but multiple times in different events of a sequence.
2. The length of a sequence: the number of instances of items in a sequence. Length (SID: 40) ?

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)>

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

Formal definition:

A sequence $\alpha = \langle a_1 a_2 \cdots a_n \rangle$ is called a **subsequence** of another sequence $\beta = \langle b_1 b_2 \cdots b_m \rangle$, and β is a **supersequence** of α, denoted as $\alpha \sqsubseteq \beta$, if there exist integers $1 \le j_1 < j_2 < \cdots < j_n \le m$ such that $a_1 \subseteq b_{j_1}$, $a_2 \subseteq b_{j_2}$, …, $a_n \subseteq b_{j_n}$. For example, if $\alpha = \langle (ab), d \rangle$ and $\beta = \langle (abc), (de) \rangle$, where a, b, c, d, and e are items, then α is a subsequence of β and β is a supersequence of α.

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*

A Basic Property of Sequential Patterns: Apriori

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

GSP (Generalized Sequential Patterns): Apriori-Based Sequential Pattern Mining

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

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

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

min_sup = 2

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

GSP (Generalized Sequential Patterns): Apriori-Based Sequential Pattern Mining

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

- Initial candidates: All 8-singleton sequences
  - <a>, <b>, <c>, <d>, <e>, <f>, <g>, <h>
- Scan DB once, count support for each candidate
- Generate length-2 candidate sequences

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 (Generalized Sequential Patterns): Apriori-Based Sequential Pattern Mining

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

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

	<a>	<b>	<c>	<d>	<e>	<f>
<a>	<aa>	<ab>	<ac>	<ad>	<ae>	<af>
<b>	<ba>	<bb>	<bc>	<bd>	<be>	<bf>
<c>	<ca>	<cb>	<cc>	<cd>	<ce>	<cf>
<d>	<da>	<db>	<dc>	<dd>	<de>	<df>
<e>	<ea>	<eb>	<ec>	<ed>	<ee>	<ef>
<f>	<fa>	<fb>	<fc>	<fd>	<fe>	<ff>

	<a>	<b>	<c>	<d>	<e>	<f>
<a>		<(ab)>	<(ac)>	<(ad)>	<(ae)>	<(af)>
<b>			<(bc)>	<(bd)>	<(be)>	<(bf)>
<c>				<(cd)>	<(ce)>	<(cf)>
<d>					<(de)>	<(df)>
<e>						<(ef)>
<f>						

Why?

GSP (Generalized Sequential Patterns): Apriori-Based Sequential Pattern Mining

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

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

	<a>	<b>	<c>	<d>	<e>	<f>
<a>	<aa>	<ab>	<ac>	<ad>	<ae>	<af>
<b>	<ba>	<bb>	<bc>	<bd>	<be>	<bf>
<c>	<ca>	<cb>	<cc>	<cd>	<ce>	<cf>
<d>	<da>	<db>	<dc>	<dd>	<de>	<df>
<e>	<ea>	<eb>	<ec>	<ed>	<ee>	<ef>
<f>	<fa>	<fb>	<fc>	<fd>	<fe>	<ff>

	<a>	<b>	<c>	<d>	<e>	<f>
<a>		<(ab)>	<(ac)>	<(ad)>	<(ae)>	<(af)>
<b>			<(bc)>	<(bd)>	<(be)>	<(bf)>
<c>				<(cd)>	<(ce)>	<(cf)>
<d>					<(de)>	<(df)>
<e>						<(ef)>
<f>						

- Without Apriori pruning:

  (8 singletons) 8*8+8*7/2 = 92 length-2 candidates
- With pruning, length-2 candidates: 36 + 15= 51

GSP Mining and Pruning

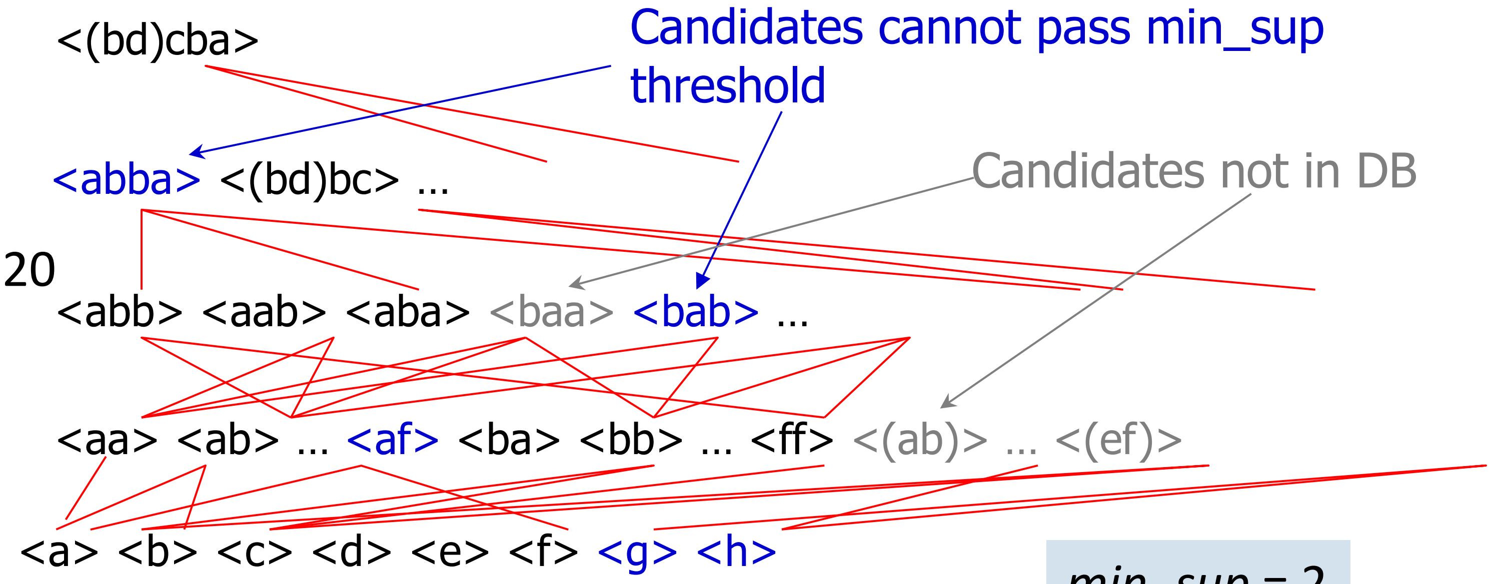
5th scan: 1 cand. 1 length-5 seq. pat.

4th scan: 8 cand. 7 length-4 seq. pat.

3rd scan: 46 cand. 20 length-3 seq. pat. 20 cand. not in DB at all

2nd scan: 51 cand. 19 length-2 seq. pat. 10 cand. not in DB at all

1st scan: 8 cand. 6 length-1 seq. pat.



min_sup = 2

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

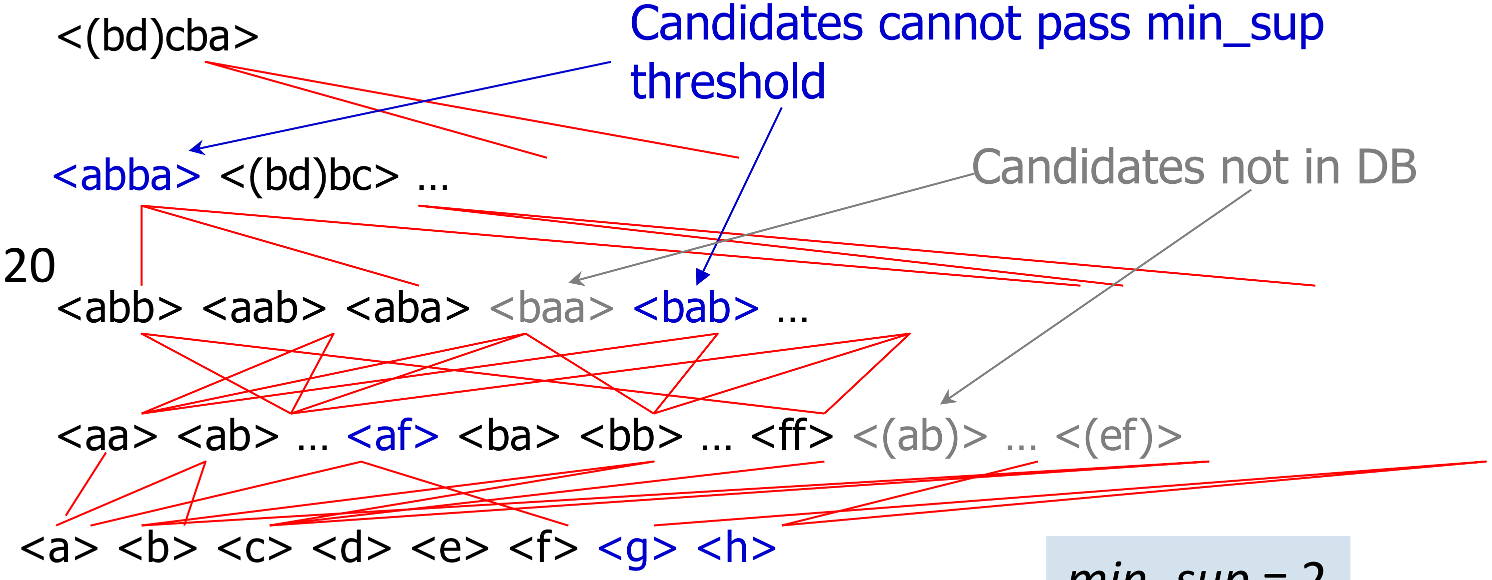
5th scan: 1 cand. 1 length-5 seq. pat.

4th scan: 8 cand. 7 length-4 seq. pat.

3rd scan: 46 cand. 20 length-3 seq. pat. 20 cand. not in DB at all

2nd scan: 51 cand. 19 length-2 seq. pat. 10 cand. not in DB at all

1st scan: 8 cand. 6 length-1 seq. pat.



min_sup = 2

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

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

GSP: Algorithm

Mining Sequential Patterns: Generalizations and Performance Improvements, Srikant and Agrawal et al. https://pdfs.semanticscholar.org/d420/ea39dc136b9e390d05e964488a65fcf6ad33.pdf

- **Phase 1:**
  - Scan over the database to identify all the frequent items, i.e., 1-element sequences
- **Phase 2:**
  - Iteratively scan over the database to discover all frequent sequences. Each iteration discovers all the sequences with the same length.
  - In the iteration to generate all *k*-sequences
  - Generate the set of all candidate *k*-sequences, C_k, by joining two (*k-1*)-sequences
    - Prune the candidate sequence if any of its *k-1* subsequences is not frequent
    - Scan over the database to determine the support of the remaining candidate sequences
  - Terminate when no more frequent sequences can be found

A detailed example illustration:

http://simpledatamining.blogspot.com/2015/03/generalized-sequential-pattern-gsp.html

Bottlenecks of GSP

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

$$1000 \times 1000 + \frac{1000 \times 999}{2} = 1{,}499{,}500$$

- Multiple scans of database in mining

- Real challenge: mining long sequential patterns
  - An exponential number of short candidates
  - A length-100 sequential pattern needs 10^{30} candidate sequences!

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

GSP: Optimization Techniques

- Applied to phase 2: computation-intensive
- Technique 1: the hash-tree data structure
  - Used for counting candidates to reduce the number of candidates that need to be checked
    - Leaf: a list of sequences
    - Interior node: a hash table
- Technique 2: data-representation transformation
  - From horizontal format to vertical format

Transaction-Time	Items
10	1, 2
25	4, 6
45	3
50	1, 2
65	3
90	2, 4
95	6

$\rightarrow$

Item	Times
1	$\rightarrow 10 \rightarrow 50 \rightarrow$ NULL
2	$\rightarrow 10 \rightarrow 50 \rightarrow 90 \rightarrow$ NULL
3	$\rightarrow 45 \rightarrow 65 \rightarrow$ NULL
4	$\rightarrow 25 \rightarrow 90 \rightarrow$ NULL
5	$\rightarrow$ NULL
6	$\rightarrow 25 \rightarrow 95 \rightarrow$ NULL
7	$\rightarrow$ NULL

SPADE

- **Problems in the GSP Algorithm**
  - Multiple database scans
  - Complex hash structures with poor locality
  - Scale up linearly as the size of dataset increases
- **SPADE: Sequential PAttern Discovery using Equivalence classes**
  - Use a vertical id-list database
  - Prefix-based equivalence classes
  - Frequent sequences enumerated through simple temporal joins
  - Lattice-theoretic approach to decompose search space
- **Advantages of SPADE**
  - 3 scans over the database
  - Potential for in-memory computation and parallelization