Apriori Algorithm, DHP and DIC

What Is Frequent Pattern Analysis?

- Frequent pattern: a pattern (a set of items, subsequences, substructures, etc.) that occurs frequently in a data set
- First proposed by Agrawal, Imielinski, and Swami [AIS93] in the context of frequent itemsets and association rule mining
- Motivation: Finding inherent regularities in data
  - What products were often purchased together?— Beer and diapers?!
  - What are the subsequent purchases after buying a PC?
  - What kinds of DNA are sensitive to this new drug?
  - Can we automatically classify web documents?
- Applications
  - Basket data analysis, cross-marketing, catalog design, sale campaign analysis, Web log (click stream) analysis, and DNA sequence analysis.

Why Is Frequent Pattern Mining Important?

- Discloses an intrinsic and important property of data sets
- Forms the foundation for many essential data mining tasks
  - Association, correlation, and causality analysis
  - Sequential, structural (e.g., sub-graph) patterns
  - Pattern analysis in spatiotemporal, multimedia, time-series, and stream data
  - Classification: associative classification
  - Cluster analysis: frequent pattern-based clustering
  - Data warehousing: iceberg cube and cube-gradient
  - Semantic data compression: fascicles
  - Broad applications

Basic Definitions

- $I = \{I_1, I_2, ..., I_m\}$, set of *items*.
- $D = \{T_1, T_2, ..., T_n\}$, *database* of *transactions*, where each transaction $T_i \subset I$. n = dbsize.
- Any non-empty subset $X \subset I$ is called an *itemset*.
- *Frequency*, *count* or *support* of an itemset X is the number of transactions in the database containing X:
  - $\text{count}(X) = |\{T_i \in D : X \subset T_i\}|$
- If count(X)/dbsize $\geq$ min_sup, some specified threshold value, then X is said to be *frequent*. ($0 \leq \text{min\_sup} \leq 1$)

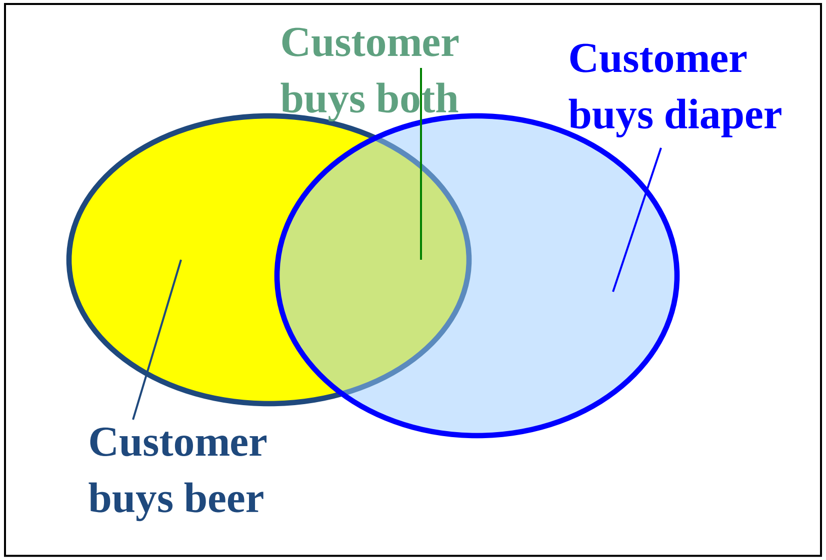
(So, $\emptyset$ is frequent automatically because count($\emptyset$) = dbsize)

Scalable Methods for Mining Frequent Itemsets

- The downward closure property (also called apriori property) of frequent itemsets
  - Any subset of a frequent itemset must be frequent
  - If **{beer, diaper, nuts}** is frequent, so is **{beer, diaper}**
  - Because every transaction having {beer, diaper, nuts} also contains {beer, diaper}
- Also (going the other way) called anti-monotonic property: any superset of an infrequent itemset must be infrequent.

Basic Concepts: Frequent Itemsets and Association Rules

Transaction-id	Items bought
10	A, B, D
20	A, C, D
30	A, D, E
40	B, E, F
50	B, C, D, E, F



- Itemset X = $\{x_1, \ldots, x_k\}$
- Find all the rules $X \Rightarrow Y$ with minimum support and confidence
  - support, *s*, probability that a transaction contains $X \cup Y$
  - confidence, *c*, conditional probability that a transaction having X also contains *Y*

Let min_sup = 50%, min_conf = 70%*

Freq. itemsets: {A:3, B:3, D:4, E:3, AD:3}

Association rules:

$A \Rightarrow D$ (60%, 100%)

$D \Rightarrow A$ (60%, 75%)

*Note that we use min_sup for both itemsets and association rules.

Support, Confidence and Lift

- Association rule is of the form $X \Rightarrow Y$, where $X, Y \subset I$ are itemsets (usually, we assume $X \cap Y = \emptyset$).
- support($X \Rightarrow Y$) = $P(X \cup Y)$ = count($X \cup Y$)/dbsize.
- confidence($X \Rightarrow Y$) = $P(Y|X)$ = count($X \cup Y$)/count(X).
- Therefore, always support($X \Rightarrow Y$) $\leq$ confidence($X \Rightarrow Y$).
- Typical values for min_sup in practical applications from 1 to 5%, for min_conf more than 50%.
- lift($X \Rightarrow Y$) = $P(Y|X)/P(Y)$

  = count($X \cup Y$)*dbsize / count(X)*count(Y),

measures the increase in likelihood of Y given X vs. random (= no info).

Apriori: A Candidate Generation-and-Test Approach

- Apriori pruning principle: If there is any itemset which is infrequent, its superset should not be generated/tested! (Agrawal & Srikant @VLDB'94 fastAlgorithmsMiningAssociationRules.pdf

  Mannila, et al. @ KDD' 94 discoveryFrequentEpisodesEventSequences.pdf
- Method:
  - Initially, scan DB once to get frequent 1-itemset
  - Generate length (k+1) candidate itemsets from length k frequent itemsets
  - Test the candidates against DB
  - Terminate when no more frequent sets can be generated

The Apriori Algorithm—An Example

min_sup = 2

Database TDB

Tid	Items
10	A, C, D
20	B, C, E
30	A, B, C, E
40	B, E

1st scan

C_1

Itemset	sup
{A}	2
{B}	3
{C}	3
{D}	1
{E}	3

L_1

Itemset	sup
{A}	2
{B}	3
{C}	3
{E}	3

C_2

Itemset
{A, B}
{A, C}
{A, E}
{B, C}
{B, E}
{C, E}

2nd scan

C_2

Itemset	sup
{A, B}	1
{A, C}	2
{A, E}	1
{B, C}	2
{B, E}	3
{C, E}	2

L_2

Itemset	sup
{A, C}	2
{B, C}	2
{B, E}	3
{C, E}	2

C_3

Itemset
{B, C, E}

3rd scan

L_3

Itemset	sup
{B, C, E}	2

The Apriori Algorithm

- <u>Pseudo-code</u>:

  C_k: Candidate itemset of size k
  L_k : frequent itemset of size k

  L_1 = {frequent items};
  for (k = 1; L_k $!=\emptyset$; k++) **do begin**
  $\boxed{C_{k+1} = \text{candidates generated from } L_k;}$
  for each transaction t in database do
  increment the count of all candidates in C_{k+1}
  that are contained in t
  L_{k+1} = candidates in C_{k+1} with min_support
  end
  return $\cup_k L_k$;

Important!
How?!
Next slide…

Important Details of Apriori

- How to generate candidates?
  - Step 1: self-joining L_k
  - Step 2: pruning
- Example of candidate-generation
  - L_3={*abc, abd, acd, ace, bcd*}
  - Self-joining: $L_3 * L_3$
    - *abcd* from *abc* and *abd*
    - *acde* from *acd* and *ace*
    - Not *abcd* from *abd* and *bcd* !

    This allows efficient implementation: sort candidates L_k lexicographically to bring together those with identical (k-1)-prefixes, …
  - Pruning:
    - *acde* is removed because *ade* is not in L_3
  - C_4={*abcd*}

How to Generate Candidates?

- Suppose the items in L_{k-1} are listed in an order
- Step 1: self-joining L_{k-1}

  insert into $\boldsymbol{C_k}$

  select $\boldsymbol{p.item_1, p.item_2, \ldots, p.item_{k-1}, q.item_{k-1}}$

  from $\boldsymbol{p \in L_{k-1}, q \in L_{k-1}}$

  where $\boldsymbol{p.item_1=q.item_1, \ldots, p.item_{k-2}=q.item_{k-2}, p.item_{k-1} < q.item_{k-1}}$
- Step 2: pruning

  forall ***itemsets c in*** $\boldsymbol{C_k}$ do

  forall ***(k-1)-subsets s of c*** do

  if *(s is not in* L_{k-1}*)* **then delete** *c* **from** C_k

How to Count Supports of Candidates?

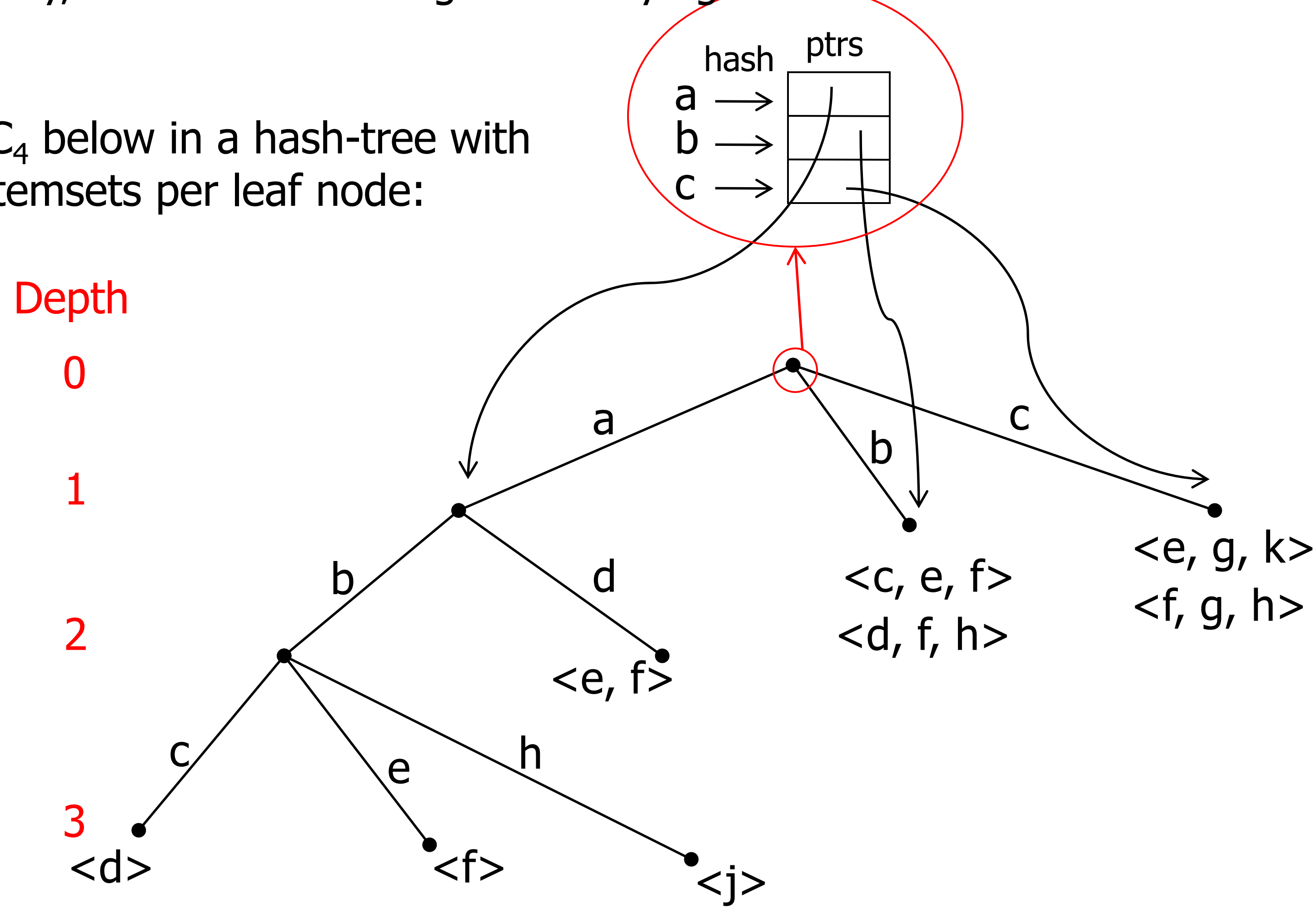
- Why counting supports of candidates a problem?
  - The total number of candidates can be very huge
  - One transaction may contain many candidates
- Method:
  - Candidate itemset C_k is stored in a *hash-tree.*
  - *Leaf node* of hash-tree contains a list of itemsets and counts.
  - *Interior node* contains a hash table keyed by items (i.e., an item hashes to a bucket) and each bucket points to a child node at next level.
  - *Subset function*: finds all the candidates contained in a transaction.
  - Increment count per candidate and return frequent ones.

Example: Using a Hash-Tree for C_k to Count Support

A hash tree is structurally same as a *prefix tree* (or *trie*), only difference being in the implementation: child pointers are stored in a *hash table* at each node in a hash tree vs. a list or array, because of the large and varying numbers of children.

Storing the C_4 below in a hash-tree with a max of 2 itemsets per leaf node:

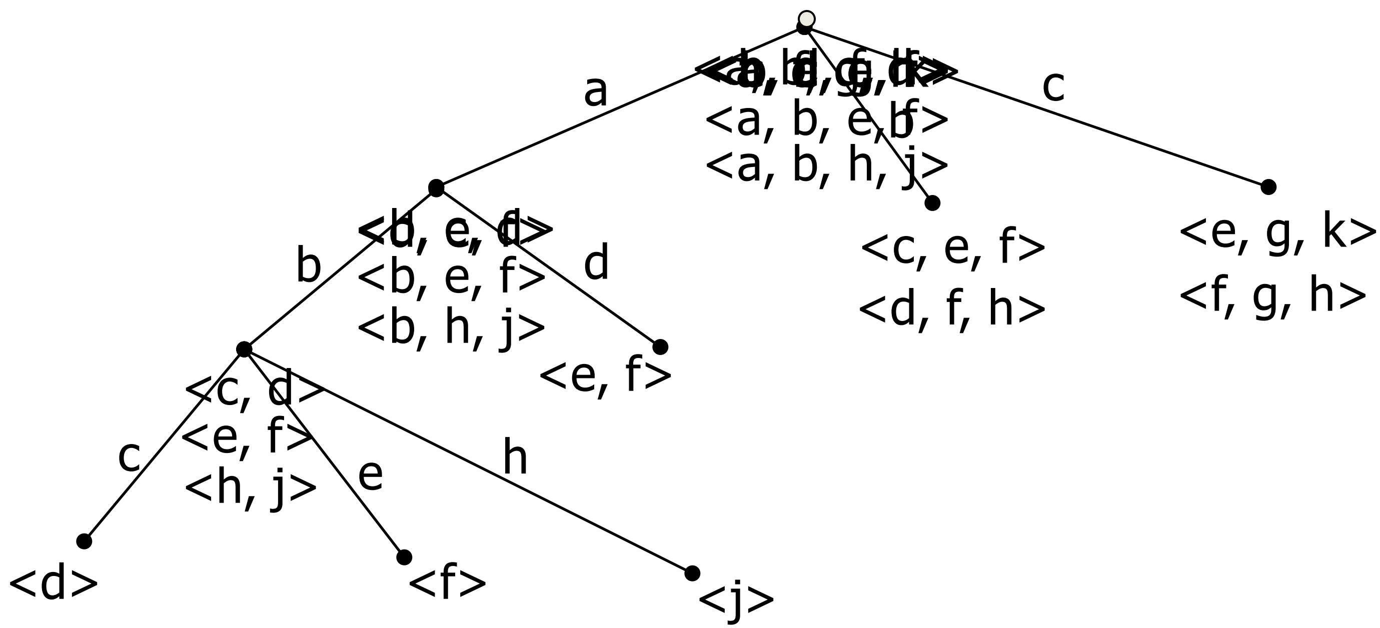
<a, b, c, d>
<a, b, e, f>
<a, b, h, j>
<a, d, e, f>
<b, c, e, f>
<b, d, f, h>
<c, e, g, k>
<c, f, g, h>



How to Build a Hash Tree on a Candidate Set

Example: Building the hash tree on the candidate set C_4 of the previous slide (*max 2 itemsets per leaf node*)

<a, b, c, d>
<a, b, e, f>
<a, b, h, j>
<a, d, e, f>
<b, c, e, f>
<b, d, f, h>
<c, e, g, k>
<c, f, g, h>



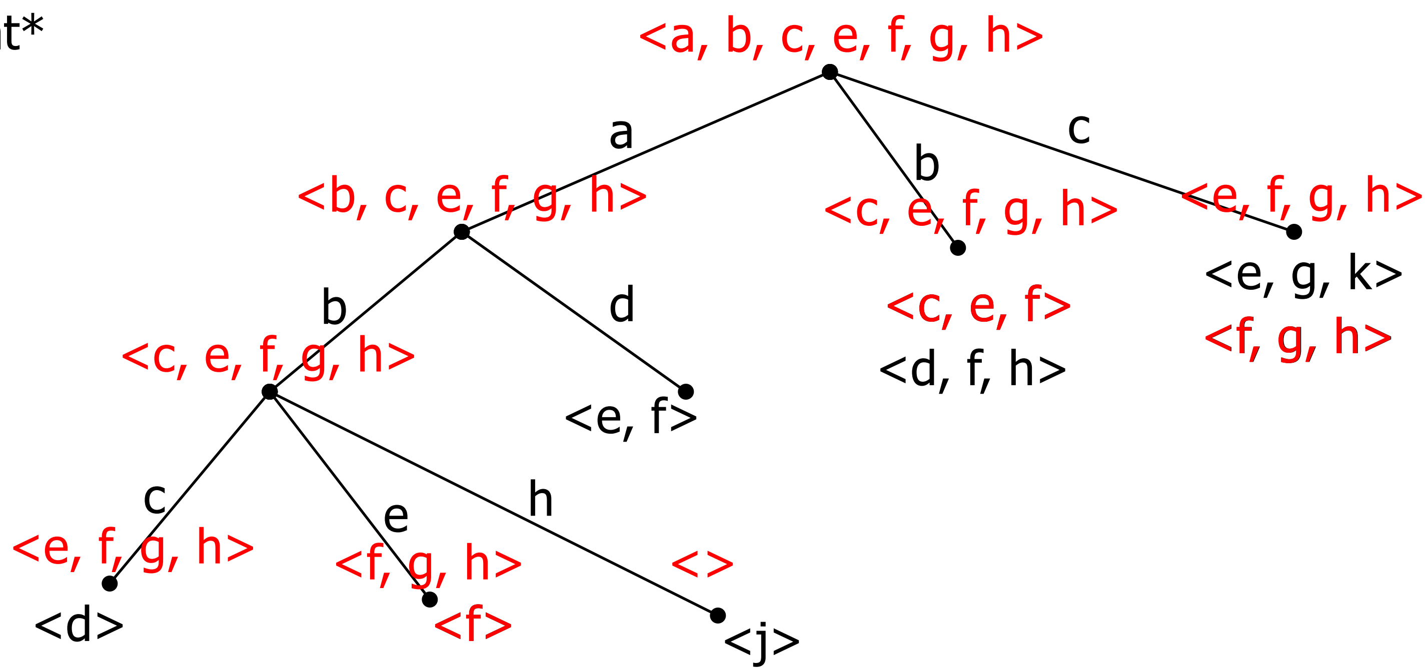
Ex: Find the candidates in C_4 contained in the transaction <a, b, c, e, f, g, h>…

How to Use a Hash-Tree for C_k to Count Support

For each transaction *T*, process *T* through the hash tree to find members of C_k contained in *T* and increment their count. After all transactions are processed, eliminate those candidates with less than min support.

Example: Find candidates in C_4 contained in *T* = <a, b, c, e, f, g, h>

C_4	Count*
<a, b, c, d>	0
<a, b, e, f>	0 1
<a, b, h, j>	0
<a, d, e, f>	0
<b, c, e, f>	0 1
<b, d, f, h>	0
<c, e, g, k>	0
<c, f, g, h>	0 1



Describe a general algorithm find candidates contained in a transaction.
Hint: Recursive…

*Counts are actually stored with the itemsets at the leaves. We show them in a separate table here for convenience.

Generating Association Rules from Frequent Itemsets

First, set min_sup for frequent itemsets to be the same as required for association rules. Pseudo-code:

for each frequent itemset l
    for each non-empty proper subset s of l
        output the rule "$s \Rightarrow l - s$" if confidence($s \Rightarrow l - s$) = (count(l)/count(s) $\geq$ min_conf

The support requirement for each output rule is automatically satisfied because:

support($s \Rightarrow l - s$) = count($s \cup (l - s)$)/dbsize = count(l)/dbsize $\geq$ min_sup

(as l is frequent). Note: Because l is frequent, so is s. Therefore, count(s) and count(l) are available (because of the support checking step of Apriori) and it's straightforward to calculate

confidence($s \Rightarrow l - s$) = count(l)/count(s).

Transactional data for an *AllElectronics* branch (Table 5.1)

TID	*List of item IDs*
T100	I1, I2, I5
T200	I2, I4
T300	I2, I3
T400	I1, I2, I4
T500	I1, I3
T600	I2, I3
T700	I1, I3
T800	I1, I2, I3, I5
T900	I1, I2, I3

Example 5.4: Generating Association Rules

Frequent itemsets from

AllElectronics database (min_sup = 0.2):

Frequent itemset	Count
{I1}	6
{I2}	7
{I3}	6
{I4}	2
{I5}	2
{I1, I2}	4
{I1, I3}	4
{I1, I5}	2
{I2, I3}	4
{I2, I4}	2
{I2, I5}	2
{I1, I2, I3}	2
{I1, I2, I5}	2

Consider the frequent itemset {I1, I2, I5}.
The non-empty proper subsets are {I1}, {I2}, {I5}, {I1, I2}, {I1, I5}, {I2, I5}.
The resulting association rules are:

Rule	Confidence
$I1 \Rightarrow I2 \wedge I5$	count{I1, I2, I5}/count{ I1} = 2/6 = 33%
$I2 \Rightarrow I1 \wedge I5$	?
$I5 \Rightarrow I1 \wedge I2$	?
$I1 \wedge I2 \Rightarrow I5$	?
$I1 \wedge I5 \Rightarrow I2$	?
$I2 \wedge I5 \Rightarrow I1$	?

How about association rules from other frequent itemsets?

Challenges of Frequent Itemset Mining

- Challenges
  - Multiple scans of transaction database
  - Huge number of candidates
  - Tedious workload of support counting for candidates
- Improving Apriori: general ideas
  - Reduce passes of transaction database scans
  - Shrink number of candidates
  - Facilitate support counting of candidates

Improving Apriori – 1

DHP: Direct Hashing and Pruning, by

J. Park, M. Chen, and P. Yu. An effective hash-based algorithm for mining association rules. In *SIGMOD'95*:

effectiveHashBasedAlgorithmMiningAssociationRules.pdf

Three Main ideas:

a. Candidates are restricted to be subsets of transactions.

E.g., if {*a*, *b*, *c*} and {*d*, *e*, *f*} are two transactions and all 6 items *a*, *b*, *c*, *d*, *e*, *f* are frequent, then Apriori considers ${}^6C_2$ = 15 candidate 2-itemsets, viz., *ab*, *ac*, *ad*, …. However, DHP considers only 6 candidate 2-itemsets, viz., *ab*, *ac*, *bc*, *de*, *df*, *ef*.

Possible downside: Have to visit transactions in the database (on disk)!

Ideas behind DHP

b. A hash table is used to count support of itemsets of small size.

E.g., hash table created using hash fn.
h(Ix, Iy) = (10x +y) mod 7
from Table 5.1

Bucket address	0	1	2	3	4	5	6
Bucket count	2	2	4	2	2	4	4
Bucket contents	{I1, I4}	{I1, I5}	{I2, I3}	{I2, I4}	{I2, I5}	{I1, I2}	{I1, I3}
	{I3, I5}	{I1, I5}	{I2, I3}	{I2, I4}	{I2, I5}	{I1, I2}	{I1, I3}
			{I2, I3}			{I1, I2}	{I1, I3}
			{I2, I3}			{I1, I2}	{I1, I3}

If min_sup = 3, the itemsets in buckets 0, 1, 3, 4, are infrequent and pruned.

Ideas behind DHP

c. Database itself is pruned by removing transactions based on the logic that a transaction can contain a frequent (*k+1*)-itemset only if contains at least *k+1* different frequent *k*-itemsets. So, a transaction that **doesn't** contain *k+1* frequent *k*-itemsets can be pruned.

E.g., say a transaction is {*a*, *b*, *c*, *d*, *e*, *f* }. Now, if it contains a frequent 3-itemset, say *aef*, then it contains the 3 frequent 2-itemsets *ae*, *af*, *ef*.

So, at the time that L_k, the frequent *k*-itemsets are determined, one can check transactions according to the condition above for possible pruning before the next stage.

Say, we have determined L_2 = {*ac*, *bd*, *eg*, *eh*, *fg* }. Then, we can drop the transaction {*a*, *b*, *c*, *d*, *e*, *f* } from the database for the next step. Why?

Improving Apriori – 2

Partition: Scanning the Database only Twice, by

Savasere, E. Omiecinski, and S. Navathe. An efficient algorithm for mining association in large databases. In VLDB'95: efficientAlgMiningAssocRulesLargeDB.pdf

Main Idea:

Partition the database (first scan) into *n* parts so that each fits in main. Observe that an itemset frequent in the whole DB (globally frequent) must be frequent in **at least one** partition (locally frequent). Therefore, collection of all locally frequent itemsets forms the global candidate set. Second scan is required to find the frequent itemsets from the global candidates.

Improving Apriori – 3

Sampling: Mining a Subset of the Database, by

H. Toivonen. Sampling large databases for association rules. In *VLDB'96: [samplingLargeDatabasesForAssociationRules.pdf]()*

Main idea:

Choose a sufficiently small random sample S of the database D as to fit in main. Find all frequent itemsets in S (locally frequent) using a lower min_sup value (e.g., 1.5% instead of 2%) to lessen the probability of missing globally frequent itemsets. With high prob: locally frequent $\supseteq$ globally frequent.

Test each locally frequent if globally frequent!

Improving Apriori – 4

S. Brin, R. Motwani, J. Ullman, and S. Tsur. Dynamic itemset counting and implication rules for market basket data. In *SIGMOD'97* :

dynamicItemSetCounting.pdf

Does this name ring a bell?!

Applying the Apriori method to a special problem

S. Guha. Efficiently Mining Frequent Subpaths. In *AusDM'09*:

efficientlyMiningFrequentSubpaths.pdf

Problem Context

Mining frequent patterns in a database of transactions

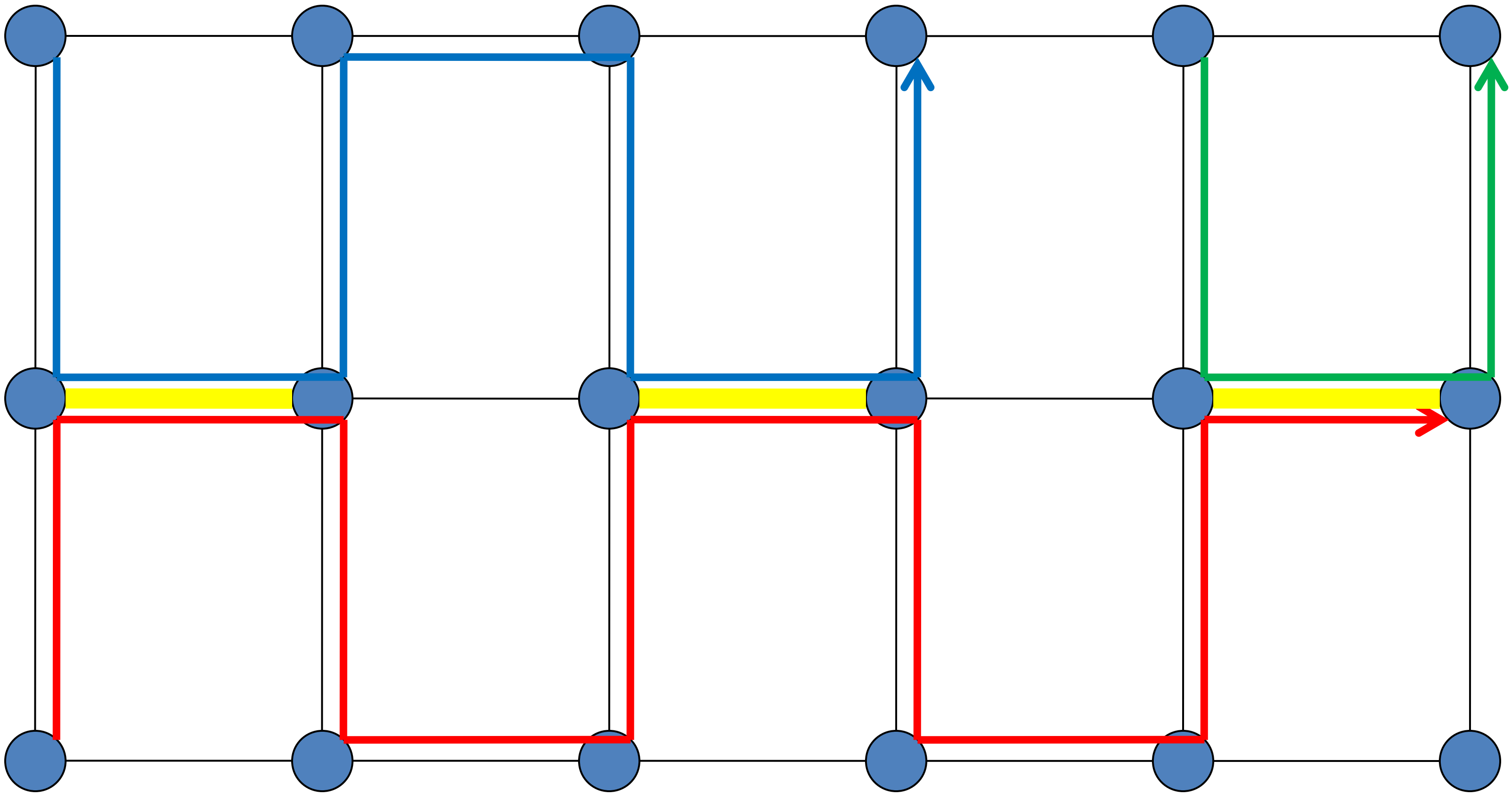
$\supseteq$

Mining frequent subgraphs in a database of graph transactions

$\supseteq$

Mining frequent subpaths in a database of path transactions in a fixed graph

Frequent Subpaths



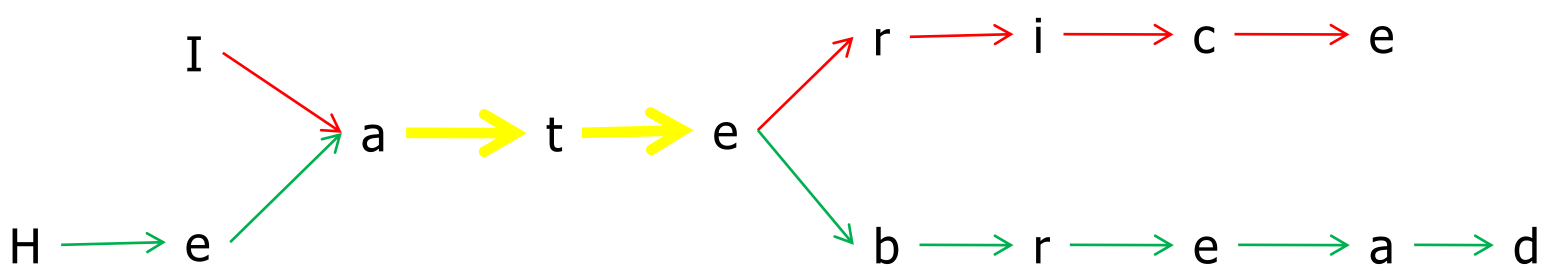
$min\_sup = 2$

Applications

- Predicting network hotspots.
- Predicting congestion in road traffic.
- Non-graph problems may be modeled as well.

E.g., finding frequent text substrings:

- I ate rice
- He ate bread



Paths in the complete graph on characters

AFS (Apriori for Frequent Subpaths)

- Code
- How it exploits the special environment of a graph to run faster than Apriori

AFS (Apriori for Frequent Subpaths)

<u>AFS</u>

L_0 = {frequent 0-subpaths};

for (k = 1; $L_{k-1} \neq \emptyset$; k++)

{

C_k = AFSextend(L_{k-1}); // Generate candidates.

C_k = AFSprune(C_k); // Prune candidates.

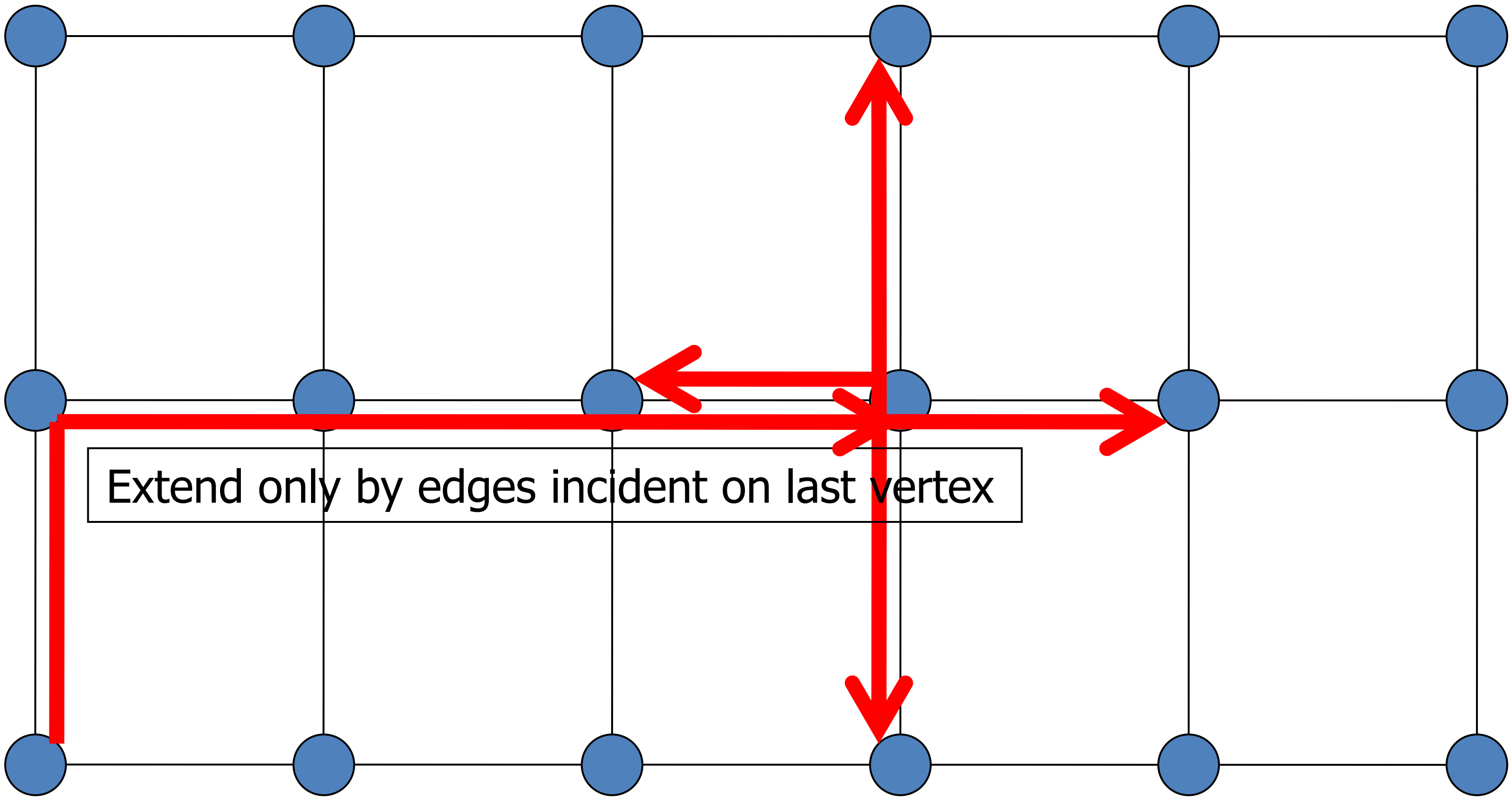
L_k = AFScheckSupport(C_k); // Eliminate candidate

// if support too low.

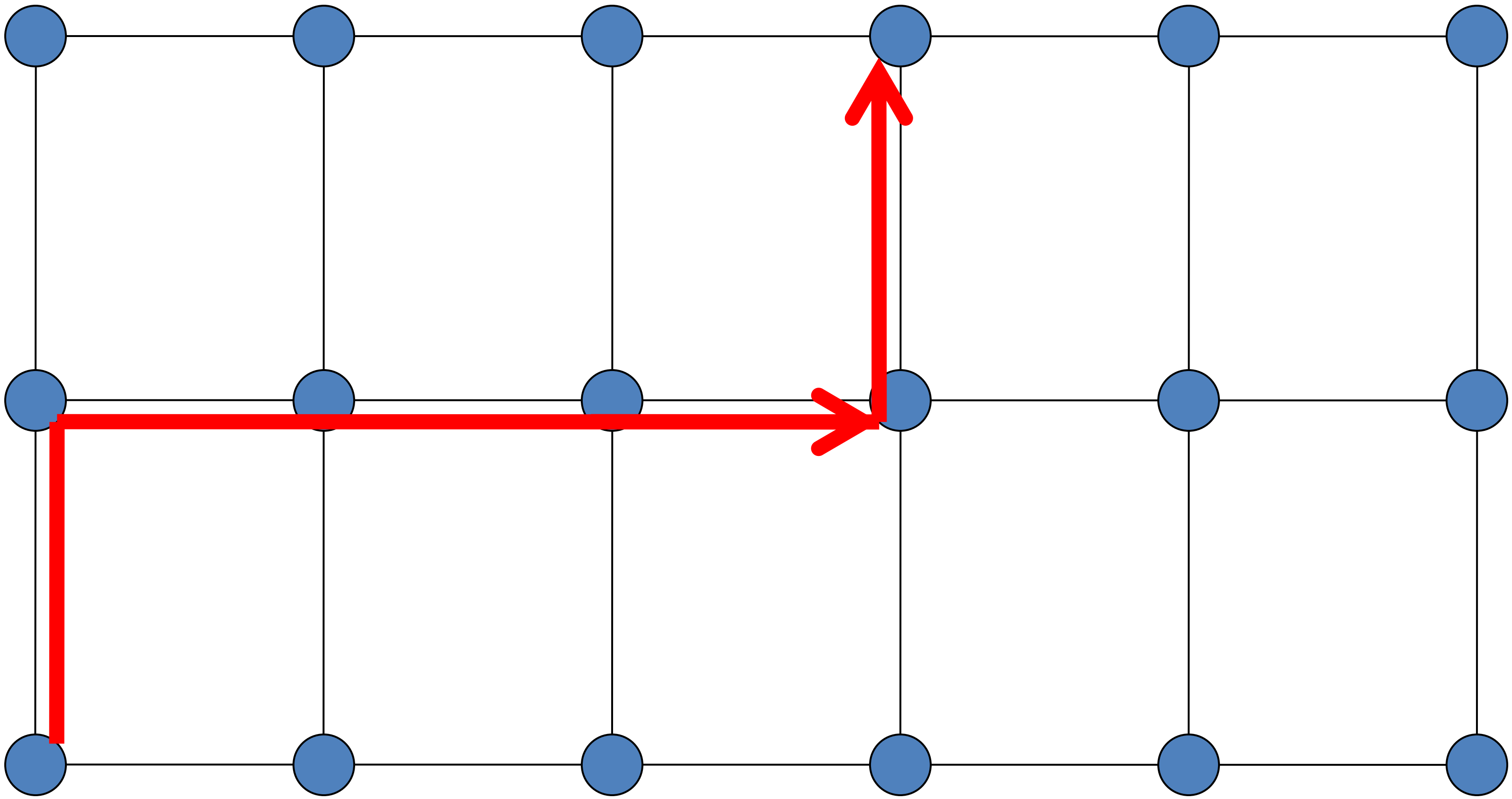
}

return $\cup_k L_k$; // Returns all frequent supaths.

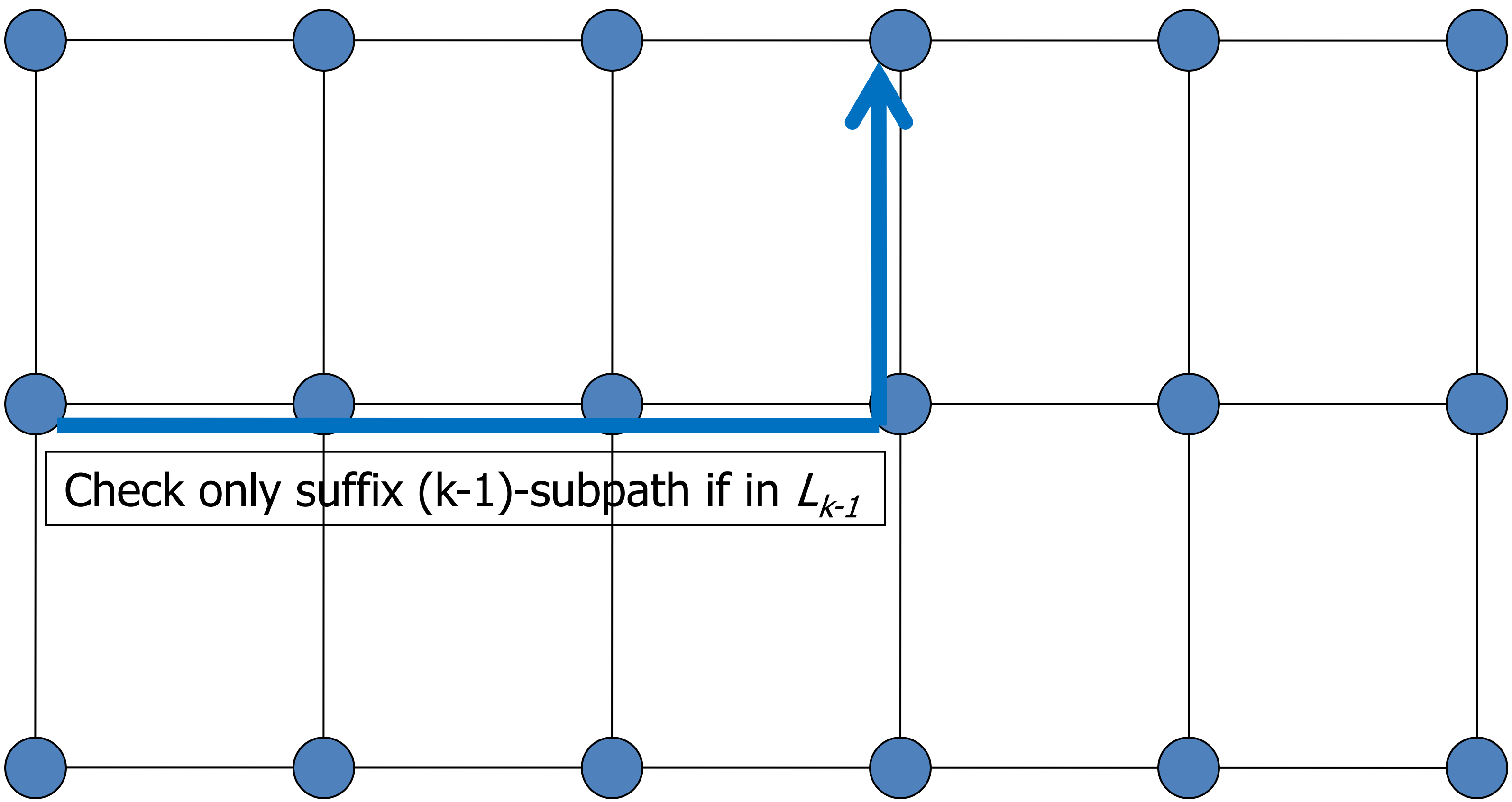
Frequent Subpaths: Extending paths (cf. Apriori joining)



Frequent Subpaths: Pruning paths (cf. Apriori pruning)



Frequent Subpaths: Pruning paths (cf. Apriori pruning)



Analysis

- The paper contains an analysis of the run-time of Apriori vs. AFS (even if you are not interested in AFS the analysis of Apriori might be useful)

A Different Approach

Determining Itemset Counts without Candidate Generation by building so-called FP-trees (FP = frequent pattern), by J. Han, J. Pei, Y. Yin. Mining Frequent Itemsets without Candidate Generation. In *SIGMOD'00:*

miningFreqPatternsWithoutCandidateGen.pdf

FP-Tree Example

A nice example of constructing an FP-tree:

FP-treeExample.pdf (note that I have annotated it)

Experimental Comparisons

A paper comparing the performance of various algorithms:

"Real World Performance of Association Rule Algorithms", by Zheng, Kohavi and Mason (KDD '01)

Mining Frequent Itemsets using Vertical Data Format

Vertical data format of the *AllElectronics* database (Table 5.1)

Min_sup = 2

Itemset	*TID_set*
I1	{T100, T400, T500, T700, T800, T900}
I2	{T100, T200, T300, T400, T600, T800, T900}
I3	{T300, T500, T600, T700, T800, T900}
I4	{T200, T400}
I5	{T100, T800}

By intersecting TID_sets.

2-itemsets in VDF

Itemset	*TID_ set*
{I1, I2}	{T100, T400, T800, T900}
{I1, I3}	{T500, T700, T800, T900}
~~{I1, I4}~~	~~{T400}~~
{I1, I5}	{T100, T800}
{I2, I3}	{T300, T600, T800, T900}
{I2, I4}	{T200, T400}
{I2, I5}	{T100, T800}
~~{I3, I5}~~	~~{T800}~~

3-itemsets in VDF

Itemset	*TID_ set*
{I1, I2, I3}	{T800, T900}
{I1, I2, I5}	{T100, T800}

By intersecting TID_sets. Optimize by using Apriori principle, e.g., no need to intersect {I1, I2} and {I2, I4} because {I1, I4} is not frequent.

Paper presenting so-called ECLAT algorithm for frequent itemset mining using VDF format:
M. Zaki (IEEE Trans. KDM '00): Scalable Algorithms for Association Mining
scalableAlgorithmsAssociationMining.pdf

Closed Frequent Itemsets and Maximal Frequent Itemsets

- A long itemset contains an exponential number of sub-itemsets, e.g., $\{a_1, \ldots, a_{100}\}$ contains $\binom{100}{1} + \binom{100}{2} + \ldots + \binom{100}{100} = 2^{100} - 1 = 1.27 \times 10^{30}$ sub-itemsets!
- *Problem*: Therefore, if there exist long frequent itemsets, then the miner will have to list an exponential number of frequent itemsets.
- *Solution*: Mine *closed frequent itemsets and/or maximal frequent itemsets* instead
- An itemset X is *closed* if X there exists *no super-itemset* $Y \supset X$, *with the same support* as X. In other words, if we add an element to X then its support will drop.

I.e., Y is strictly bigger than X.

- X is said to be *closed frequent* if it is both closed and frequent.
- An itemset X is a *maximal frequent* if X is frequent and there exists no frequent super-itemset $Y \supset X$.
- Closed frequent itemsets give support information about all frequent itemsets, maximal frequent itemsets do not.

Examples

- DB:

T1: a, b, c

T2: a, b, c, d

T3: c, d

T4: a, e

T5: a, c

1. Find the closed sets.
2. Assume min_sup = 2, find closed frequent and max frequent sets.

Condition for an itemset to be closed

Lemma 1: Itemset I is closed if and only if for every element x $\in$ I there exists a transaction T s.t. I $\subset$ T and x $\in$ T.

Proof:

Suppose I is closed and x $\in$ I. Then, by definition, the support of I U {x} is less than the support of I. Therefore, there is at least one transaction T containing I but not containing I U {x}, which means I $\subset$ T and x $\in$ T.

Conversely, suppose that for every element x $\in$ I there exists a transaction T s.t. I $\subset$ T and x $\in$ T. It is easy to see that this means the support of any itemset strictly bigger than I is less than that of I, which means I is closed.

Intersection of closed sets is closed

Lemma 2: The intersection of any two closed itemsets A and B is closed.

Proof:

Suppose, if possible, that A and B are closed but $A \cap B$ is not closed. Since $A \cap B$ is not closed, by the (contrapositive of the) previous lemma there must exist an element $x \in A \cap B$, s.t. every transaction containing $A \cap B$ also contains x.

Since every transaction containing $A \cap B$ contains x, every transaction containing A also contains x. But this means $x \in A$, otherwise we violate the condition of the previous lemma for A to be closed.

By the same reasoning we must have $x \in B$. But then $x \in A \cap B$ which contradicts the statement above that $x \in A \cap B$. Therefore, $A \cap B$ must be closed.

Corollary: The intersection of any two closed frequent itemsets A and B is closed frequent. *Because the intersection of two frequent sets is frequent*.

Corollary: The intersection of any finite number of closed frequent itemsets is closed frequent.

Every frequent itemset is contained in a closed frequent itemset

Lemma 3: Any frequent itemset A is contained in a closed frequent itemset with the same support as A.

Proof:

Suppose A is a frequent itemset. If A is closed itself there is nothing more to prove.

So, suppose A is not closed. By definition then, there exists an x $\in$ A s.t.

A U {x} has the same support as A. If A U {x} is closed, then A U {x} is the closed frequent itemset containing A with the same support.

If A U {x} is not closed, then we can repeat the process to add another element y $\in$ A s.t. A U {x, y} has the same support as A. Again, if

A U {x, y} is closed then we are done.

If A U {x, y} is not closed, repeat the process of adding new elements until it ends – it must end because there are only a finite number of elements – when we will indeed have a closed frequent itemset containing A with the same support.

Finding the support of all frequent itemsets from the support of closed frequent itemsets

Theorem: Every frequent itemset A is contained in a unique smallest closed frequent itemset, which has the same support as A.

Proof:

From Lemma 3 we know that there is at least one closed frequent itemset containing A. Now, consider the intersection of *all* closed frequent itemsets containing A. Call this set A'. Then, by a corollary to Lemma 2, A' is also closed frequent. Moreover, A' is the smallest closed frequent itemset containing A, because it is contained in every closed frequent itemset containing A (being their intersection).

By Lemma 3, there is a closed frequent itemset, call it A'', s.t.,

support(A) = support(A''). But, we have $A \subset A' \subset A''$, because A' is smallest closed frequent itemset containing A, which means

support(A) $\geq$ support(A') $\geq$ support(A'').

Since support(A) = support(A''), we conclude that support(A) = support(A'), finishing the proof.

Finding the support of frequent itemsets from the support of closed frequent itemsets

The previous theorem allows us to find to find the support of all frequent itemsets just from knowing the support of closed frequent itemsets.

It means ambiguous situations like the following *cannot* happen:

{a, b} is frequent, and the only closed frequents sets are {a, b, c, d} with support 4 and {a, b, e, f} with support 5. So, is the support of {a, b} 4 or 5?

Why can't such a situation happen?!

Examples

- Exercise. DB = $\{\langle a_1, \ldots, a_{100}\rangle, \langle a_1, \ldots, a_{50}\rangle\}$
  - Say min_sup = 1 (absolute value, or we could say 0.5).
- What is the set of closed frequent itemsets?
  - $\langle a_1, \ldots, a_{100}\rangle$: 1
  - $\langle a_1, \ldots, a_{50}\rangle$: 2
- What is the set of maximal frequent itemsets?
  - $\langle a_1, \ldots, a_{100}\rangle$: 1
- Now, consider if $\langle a_2, a_{45}\rangle$ and $\langle a_8, a_{55}\rangle$ are frequent and what are their counts from (a) knowing maximal frequent itemsets, and (b) knowing closed frequent itemsets.

Mining Closed Frequent Itemsets Papers

– Pasquier, Bastide, Taouil, Lakhal (ICDT'99): Discovering Closed Frequent Itemsets for Association Rules

discoveringFreqClosedItemsetsAssocRules.pdf

The original paper: nicely done theory, not clear if algorithm is practical.

– Pei, Han, Mao (DMKD'00): CLOSET: An Efficient Algorithm for Mining Frequent Closed Itemset

CLOSETminingFrequentClosedItemsets.pdf

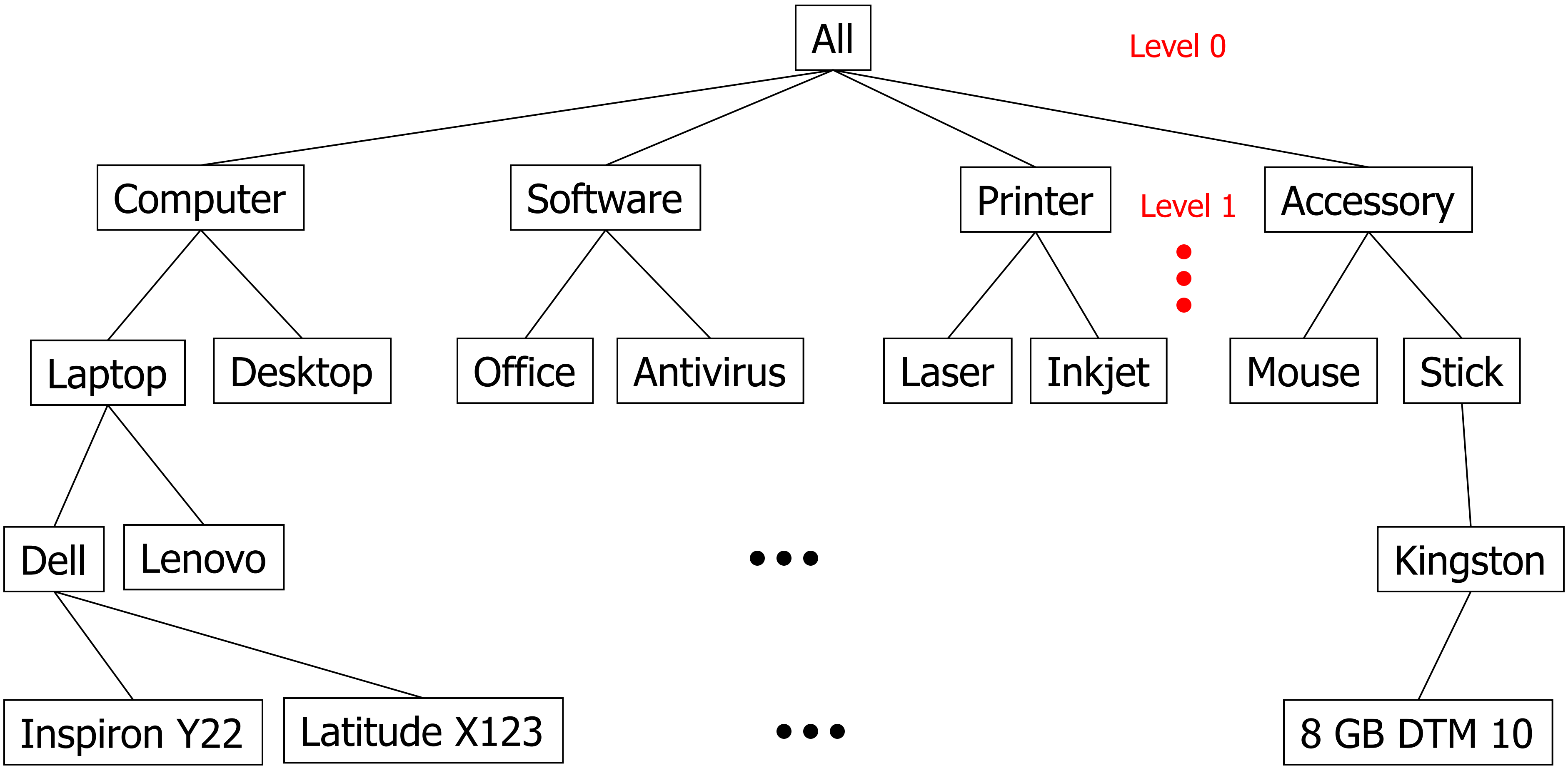
Based on FP-growth. Similar ideas (same authors).

– Zaki, Hsiao (SDM'02): CHARM: An Efficient Algorithm for Closed Itemset Mining

CHARMefficientAlgorithmClosedItemsetMining.pdf

Based on Zaki's (IEEE Trans. KDM '00) ECLAT algorithm for frequent itemset mining using the VDF format.

Mining Multilevel Association Rules



5-level concept heirarchy

Principle: Association rules at low levels may have little support; conversely, there may exist stronger rules at higher concept levels.

Multidimensional Association Rules

- Single-dimensional association rule uses a single predicate, e.g.,

  buys(X, "digital camera") $\Rightarrow$ *buys(X, "HP printer")*
- Multidimensional association rule uses multiple predicates, e.g.,

  age(X, "20…29") AND *occupation(X, "student")* $\Rightarrow$ *buys(X, "laptop")*

and

age(X, "20…29") AND *buys(X, "laptop")* $\Rightarrow$ *buys(X, "HP printer")*

Association Rules for Quantitative Data

- Quantitative data cannot be mined per se:
  - E.g., if income data is quantitative it can have values 21.3K, 44.9K, 37.3K. Then, a rule like

    income(X, 37.3K) $\Rightarrow$ *buys(X, laptop)*

    will have little support (also what does it mean? How about someone with income 37.4K?)
- However, quantitative data can be *discretized* into finite ranges, e.g., income 30K-40K, 40K-50K, etc.
  - E.g., the rule

    income(X, "30K…40K") $\Rightarrow$ *buys(X, laptop)*

    is meaningful and useful.

Checking Strong Rules using Lift

Consider:

- 10,000 transactions
- 6000 transactions included computer games
- 7500 transactions included videos
- 4000 included both computer games and videos
- min_sup = 30%, min confidence = 60%

One rule generated will be:

buys(X, "computer games") ⇒ buys(X, videos) support=40%, conf = 66%

However,

prob(buys(X, videos)) = 75%

so buying a computer game actually *reduces* the chance of buying a video!

This can be detected by checking the *lift* of the rule, viz.,

lift(computer games ⇒ videos) = 8/9 < 1.

A useful rule must have lift > 1.