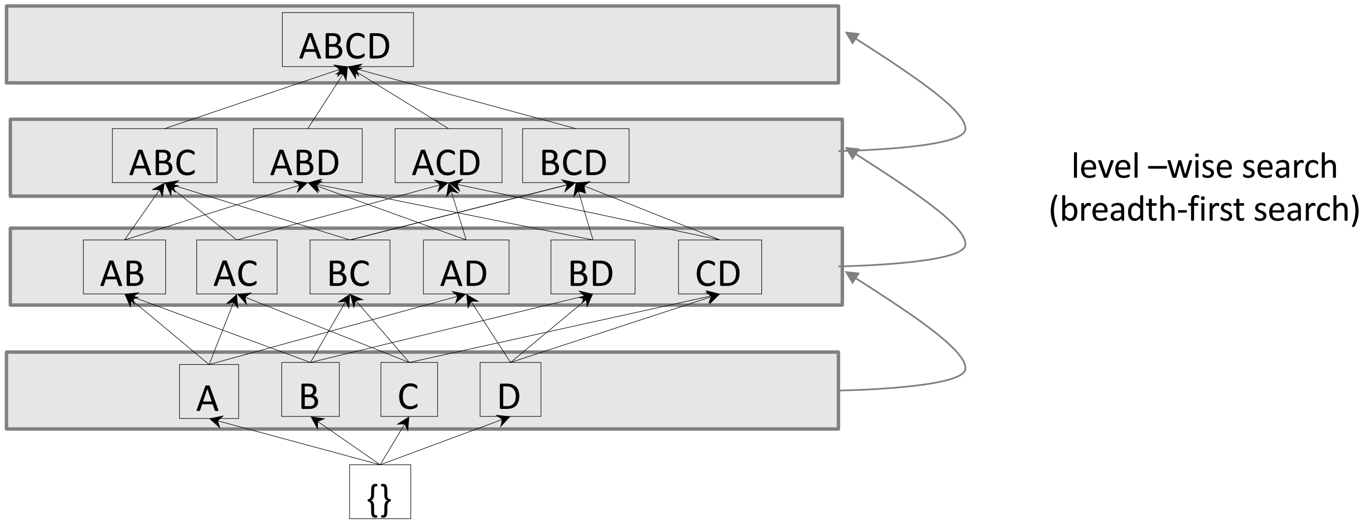
Outline

- Introduction
- Basic concepts
- Frequent Itemsets Mining (FIM) – Apriori
- Association Rules Mining

Apriori algorithm [Agrawal & Srikant @VLDB'94]

- Idea: First determine frequent 1-itemsets, then frequent 2-itemsets and so on



- Method overview:

1. Initially, scan *DB* once to get frequent 1-itemset
2. Generate length (*k*+1) candidate itemsets from length *k* frequent itemsets
3. Test the candidates against DB (one scan)
4. Repeat from Step 2. Terminate when no frequent or candidate set can be generated

Apriori property

- Naïve approach: Count the frequency of all *k*-itemsets *X* from *I*
  - generate $\sum_{k=1}^{|I|} \binom{|I|}{k} = 2^{|I|} - 1$ itemsets, i.e., $O(2^{|I|})$.
  - for each candidate itemset *X*, the algorithm evaluates whether *X* is frequent

    ➔ To reduce complexity, the set of candidates should be as small as possible!!!

- Downward closure property / Monotonic property/Apriori property of frequent itemsets:
  - If *X* is *frequent*, all its subsets Y $\subseteq$ *X* are also *frequent*.
    - e.g., if {beer, diaper, nuts} is frequent, so is {beer, diaper}
      - i.e., every transaction having {beer, diaper, nuts} also contains {beer, diaper}
    - similarly for {diaper, nuts}, {beer, nuts}

{beer, diaper, nuts} 2

{beer, diaper} ≥2

minSupport =2

  - On the contrary: When *X* is *not frequent*, all its supersets are *not frequent* and thus they should not be generated/ tested!!! → reduce the candidate itemsets set
    - e.g., if {beer, diaper} is not frequent, {beer, diaper, nuts} would not be frequent also

{beer, diaper, nuts} <=1

{beer, diaper} 1

minSupport =2

Illustration of the Apriori property

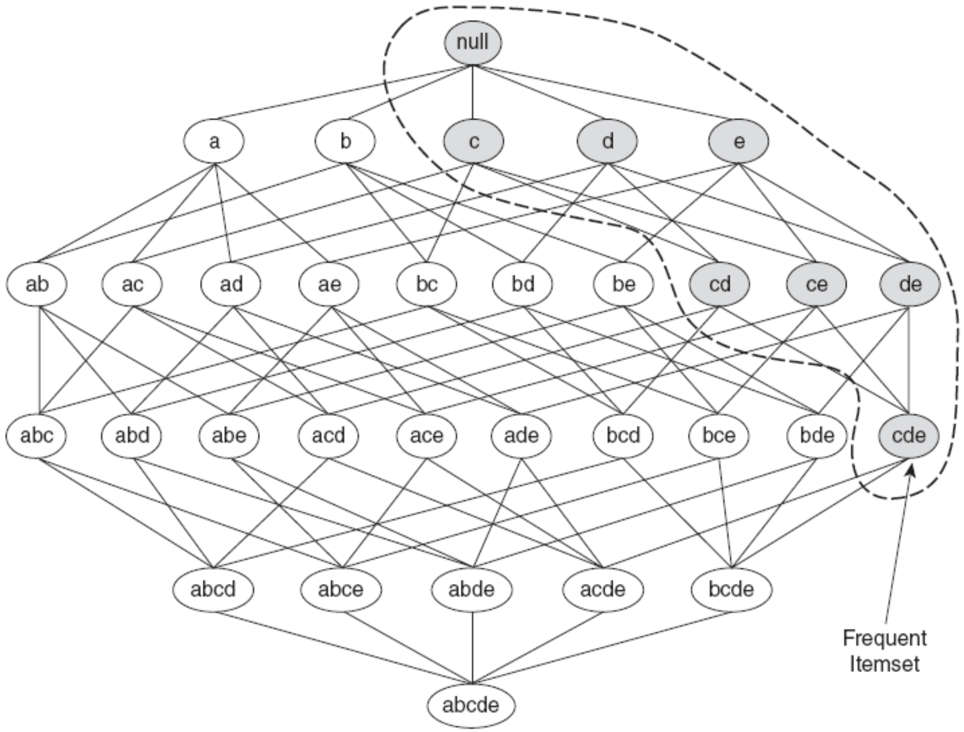


Figure 6.3. An illustration of the *Apriori* principle. If $\{c, d, e\}$ is frequent, then all subsets of this itemset are frequent.

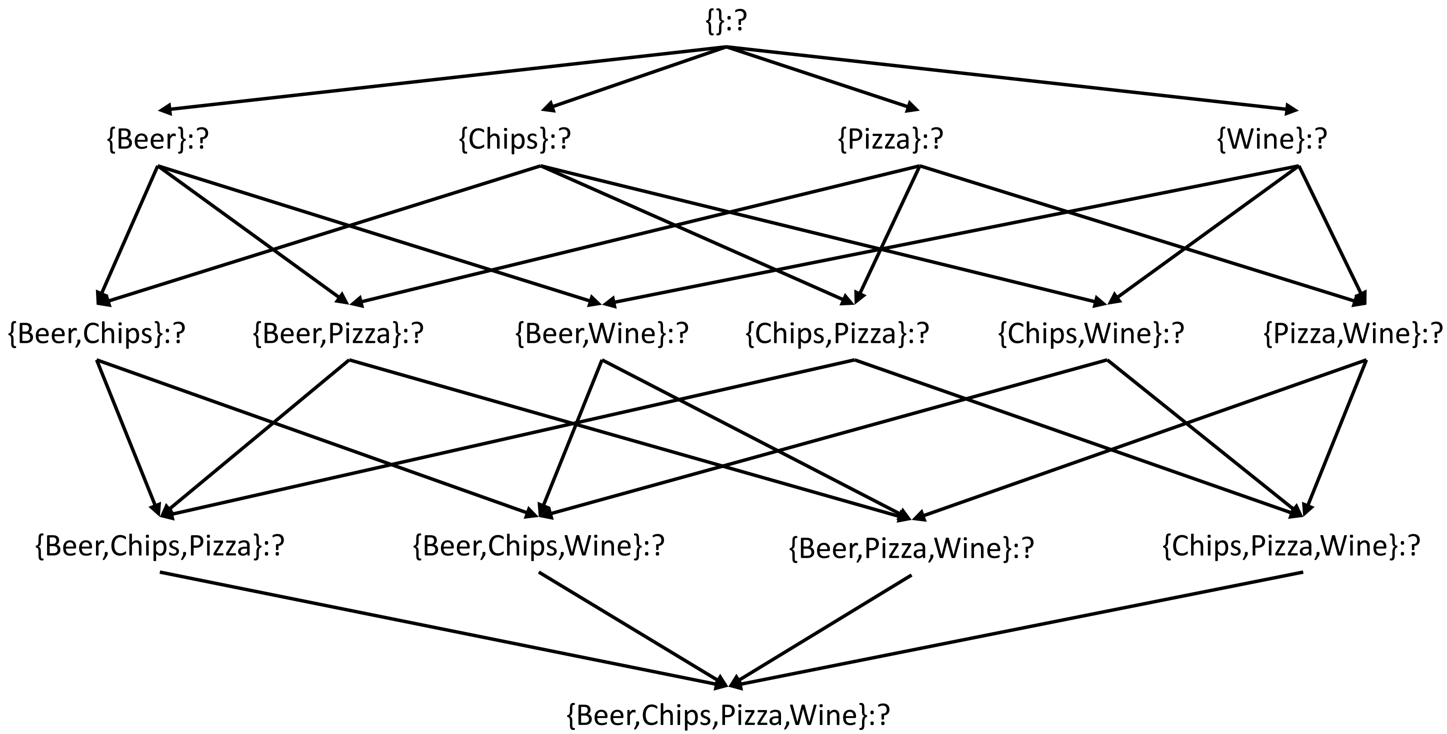
Search space and pruning

- Let us consider the following transaction database

Transaction Database
{Chips, Pizza}
{Beer, Chips}
{Chips, Pizza, Wine}
{Wine}

- and a minSupport threshold *minSupp = 2*

Search space and pruning

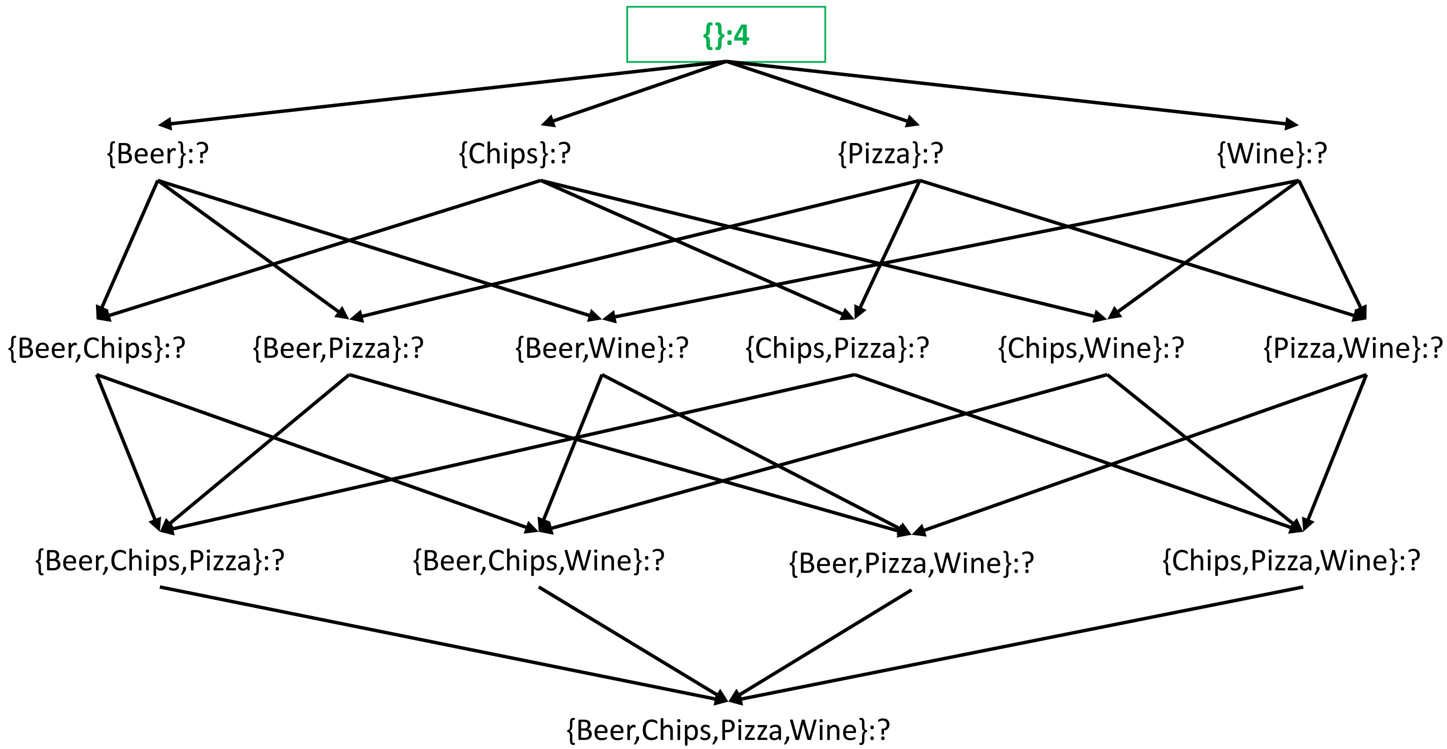


Transaction Database

{Chips, Pizza}
{Beer, Chips}
{Chips, Pizza, Wine}
{Wine}

minSupp = 2

Search space and pruning

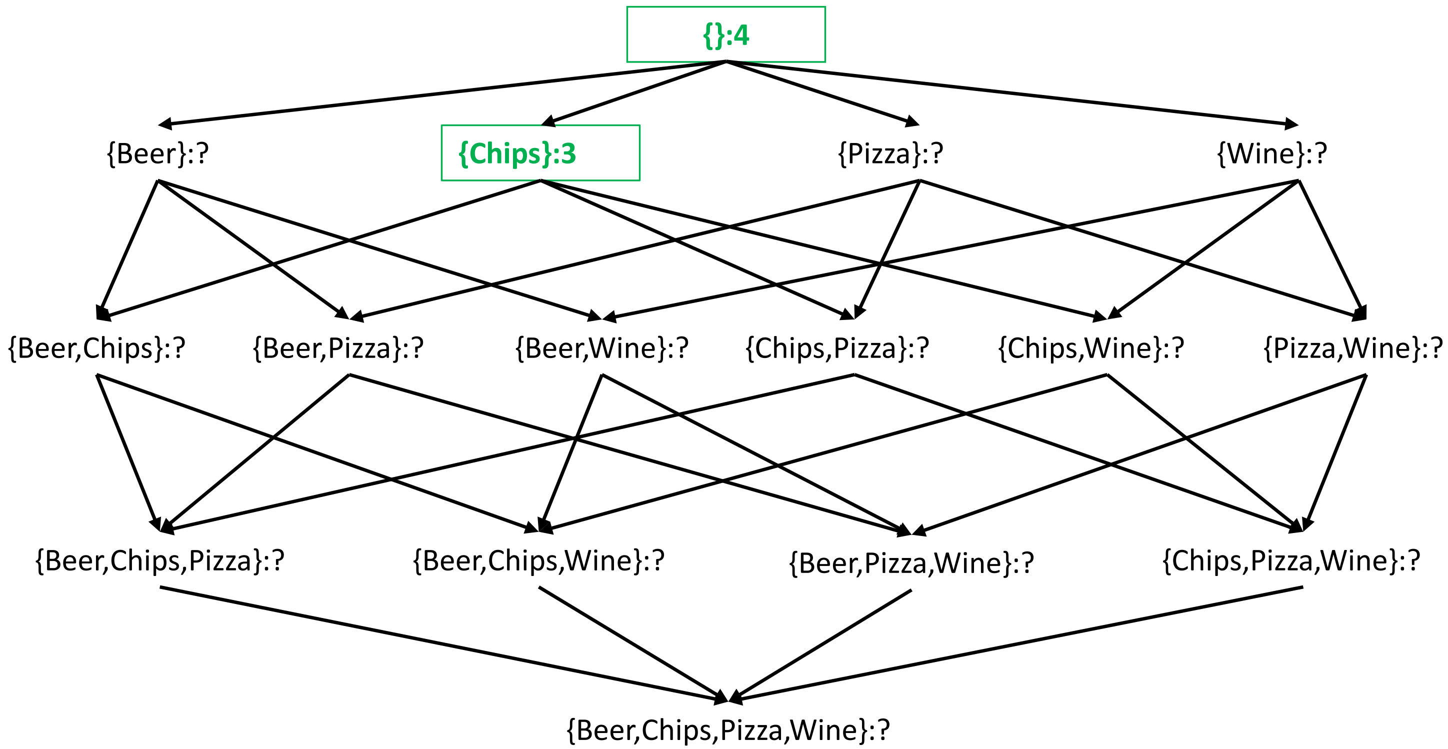


Transaction Database

{Chips, Pizza}
{Beer, Chips}
{Chips, Pizza, Wine}
{Wine}

minSupp = 2

Search space and pruning

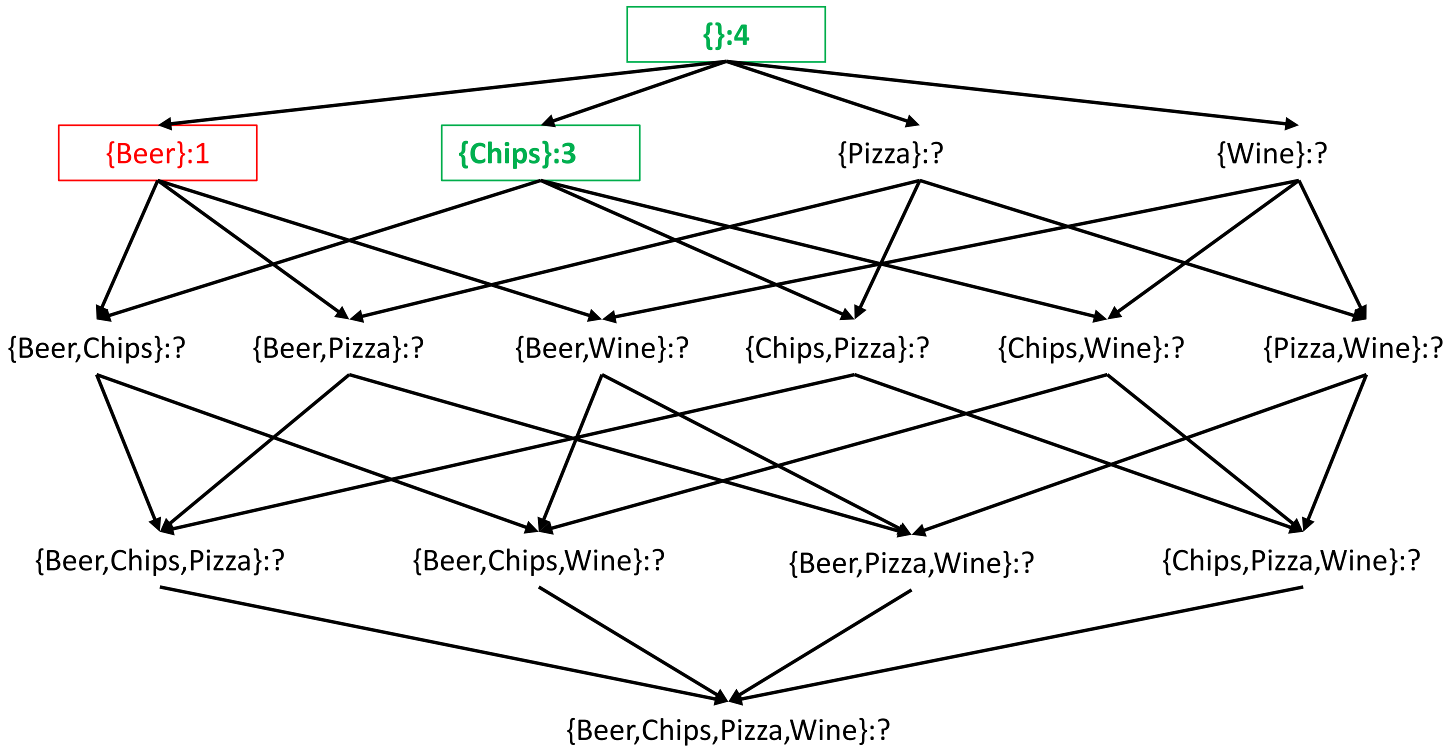


Transaction Database

{Chips, Pizza}
{Beer, Chips}
{Chips, Pizza, Wine}
{Wine}

minSupp = 2

Search space and pruning

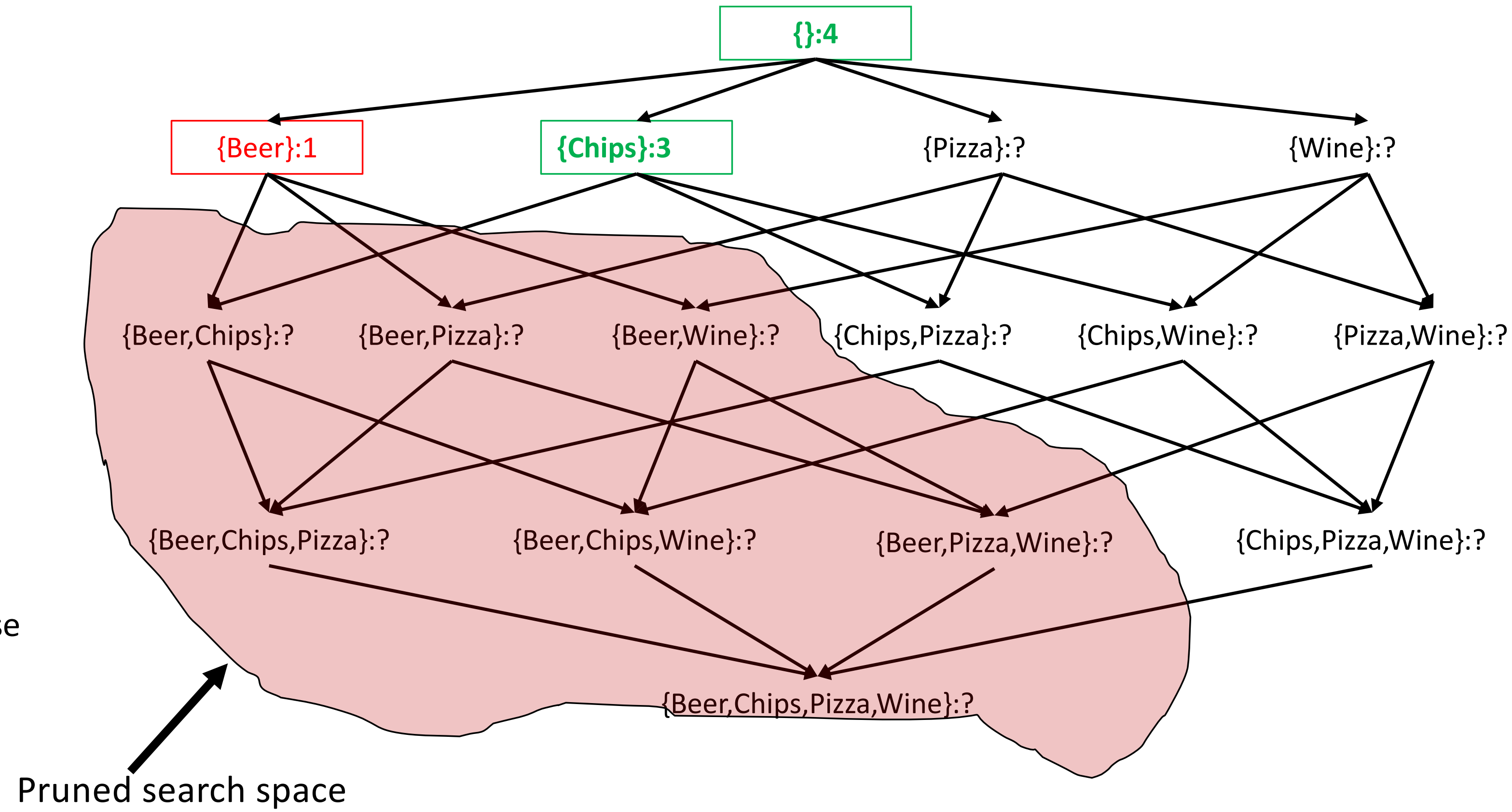


Transaction Database

{Chips, Pizza}
{Beer, Chips}
{Chips, Pizza, Wine}
{Wine}

minSupp = 2

Search space and pruning

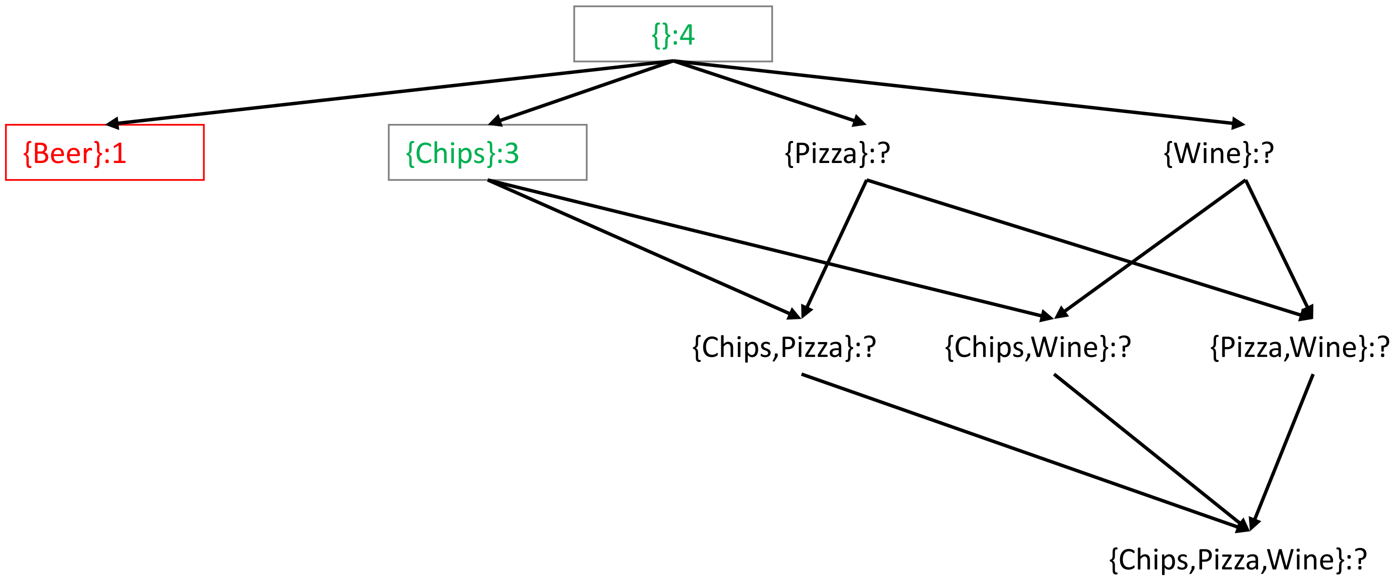


Transaction Database

{Chips, Pizza}
{Beer, Chips}
{Chips, Pizza, Wine}
{Wine}

minSupp = 2

Search space and pruning

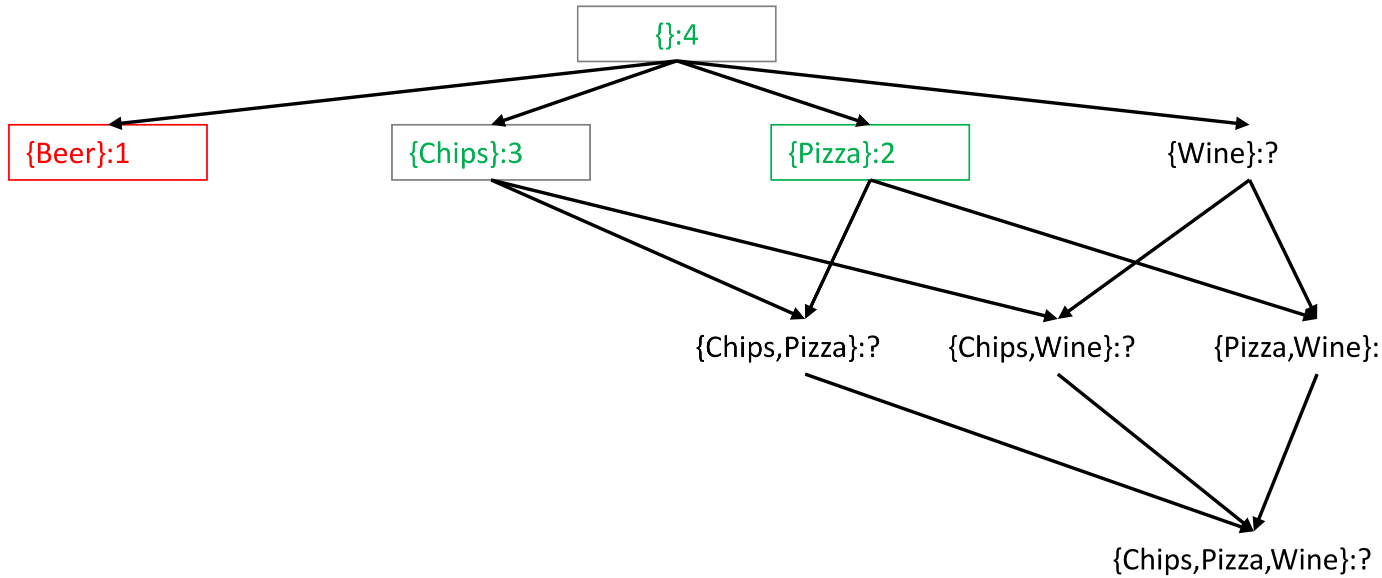


Transaction Database

{Chips, Pizza}
{Beer, Chips}
{Chips, Pizza, Wine}
{Wine}

minSupp = 2

Search space and pruning

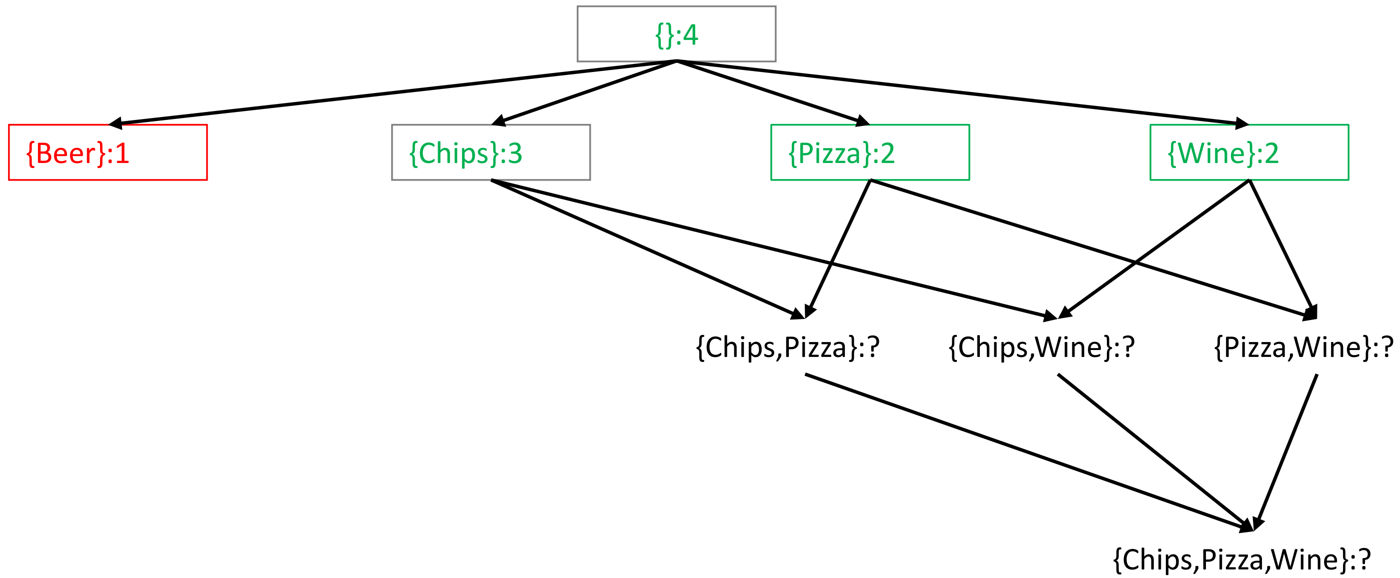


Transaction Database

{Chips, Pizza}
{Beer, Chips}
{Chips, Pizza, Wine}
{Wine}

minSupp = 2

Search space and pruning

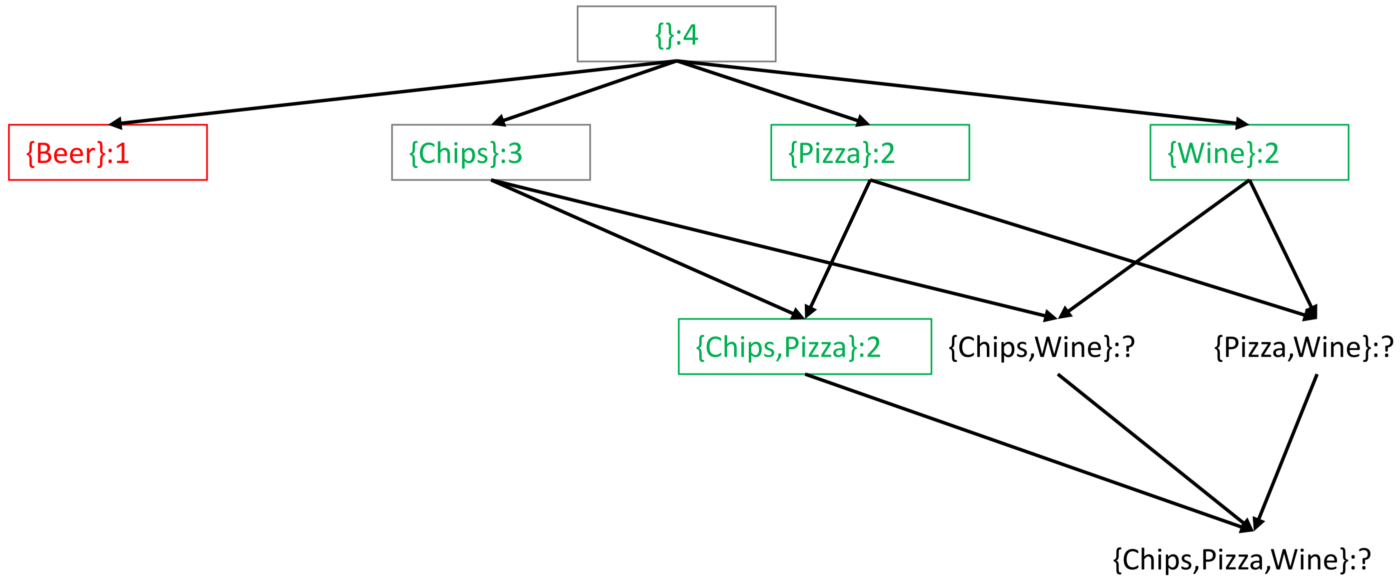


Transaction Database

{Chips, Pizza}
{Beer, Chips}
{Chips, Pizza, Wine}
{Wine}

minSupp = 2

Search space and pruning

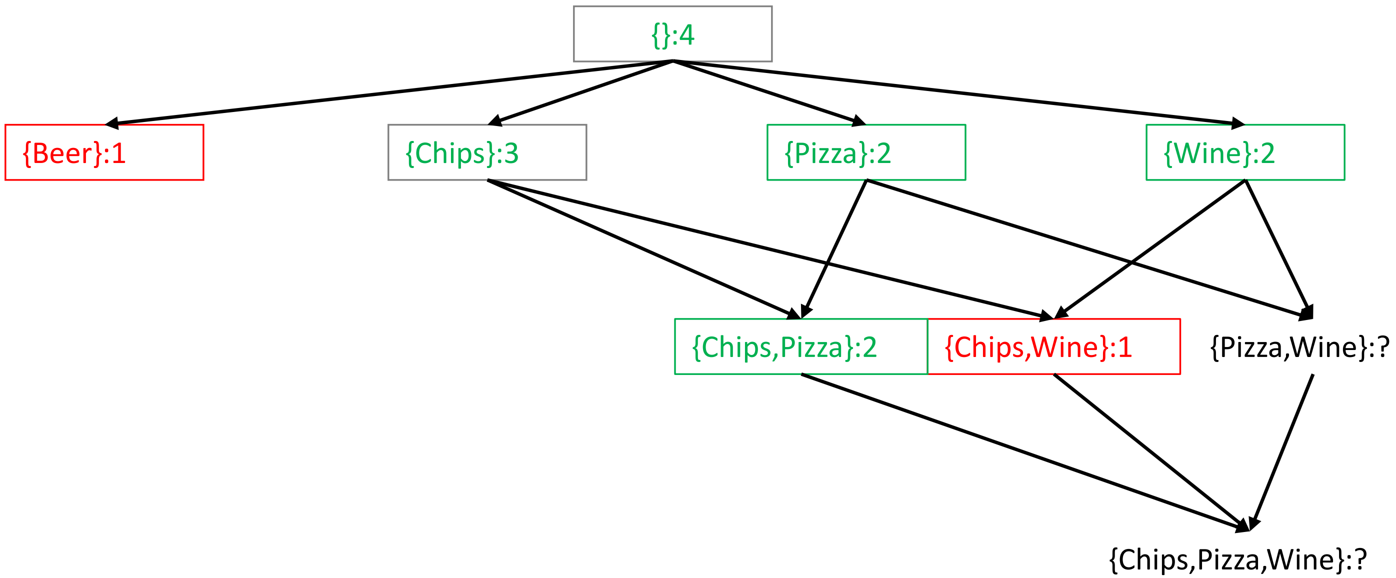


Transaction Database

{Chips, Pizza}
{Beer, Chips}
{Chips, Pizza, Wine}
{Wine}

minSupp = 2

Search space and pruning

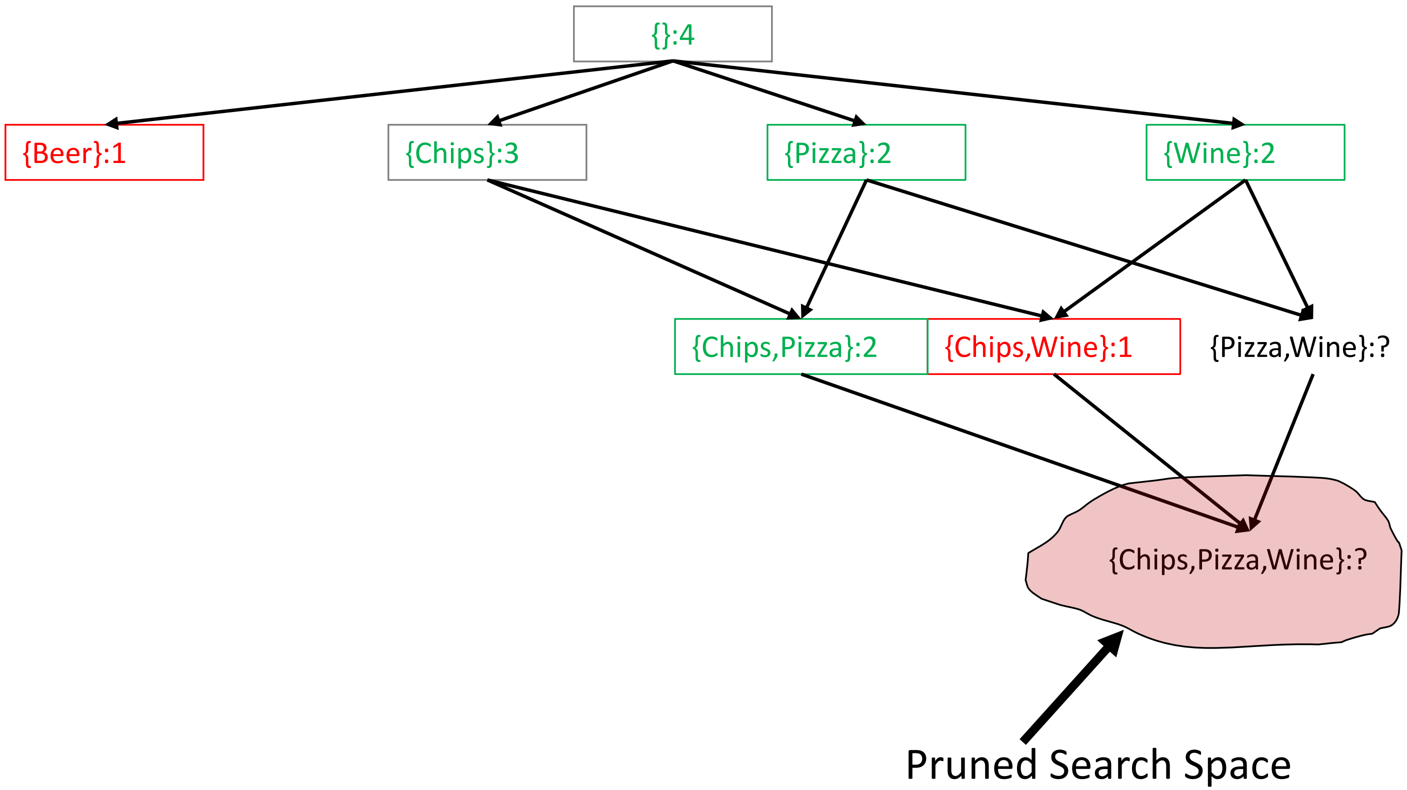


Transaction Database

{Chips, Pizza}
{Beer, Chips}
{Chips, Pizza, Wine}
{Wine}

minSupp = 2

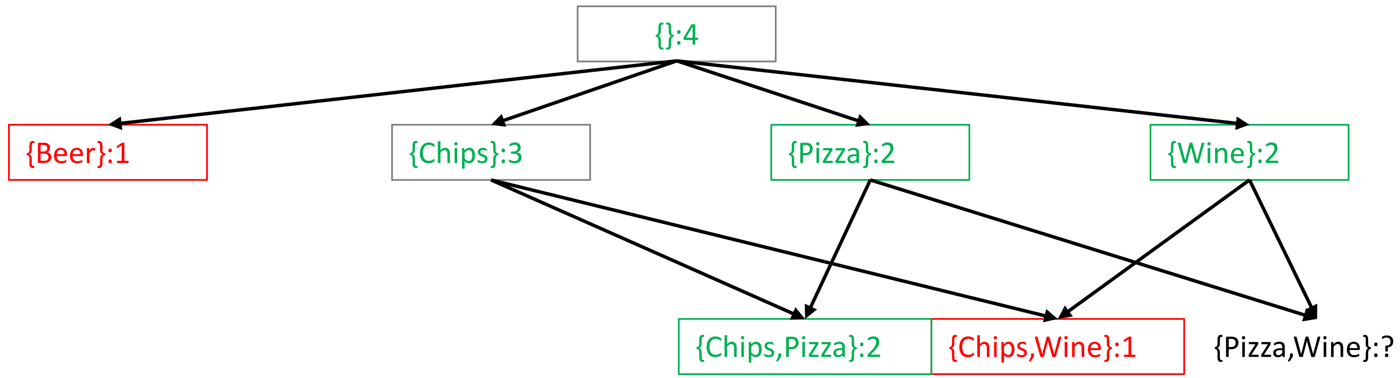
Search space and pruning



Transaction Database
{Chips, Pizza}
{Beer, Chips}
{Chips, Pizza, Wine}
{Wine}

minSupp = 2

Search space and pruning

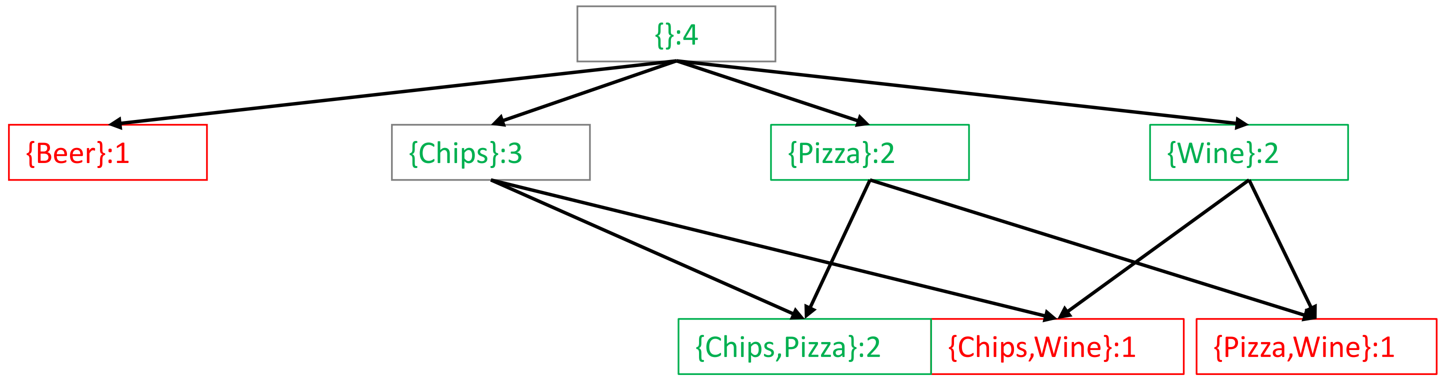


Transaction Database

{Chips, Pizza}
{Beer, Chips}
{Chips, Pizza, Wine}
{Wine}

minSupp = 2

Search space and pruning



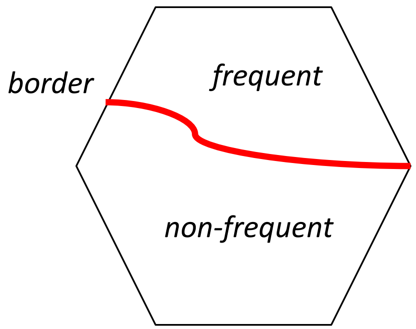
Transaction Database

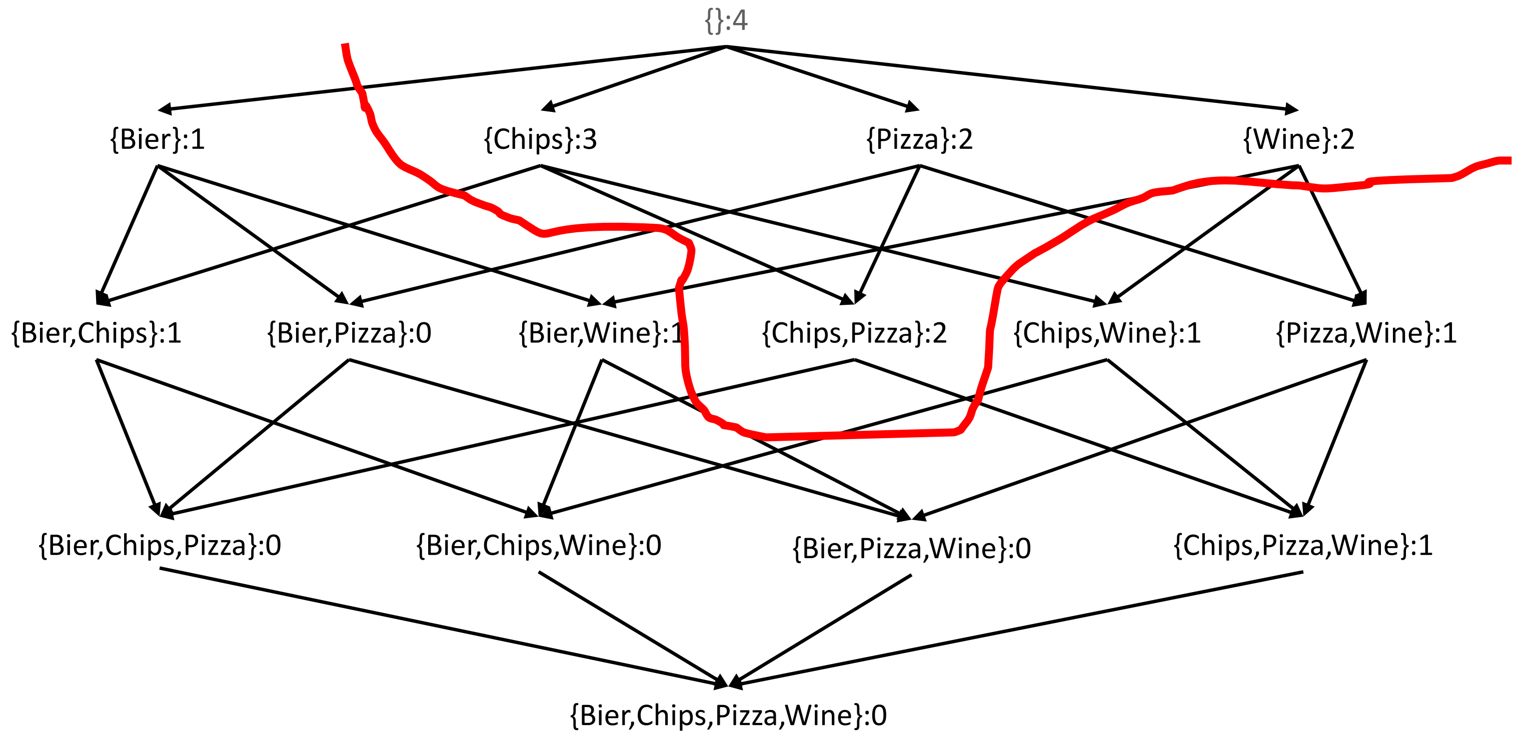
{Chips, Pizza}
{Beer, Chips}
{Chips, Pizza, Wine}
{Wine}

minSupp = 2

Search space and pruning

- Border itemsets *X*: all subsets Y ⊂ X are frequent, all supersets Z ⊃ X are not frequent





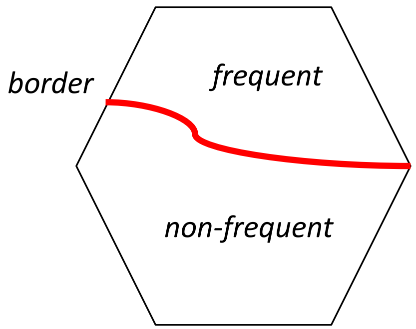
Transaction Database

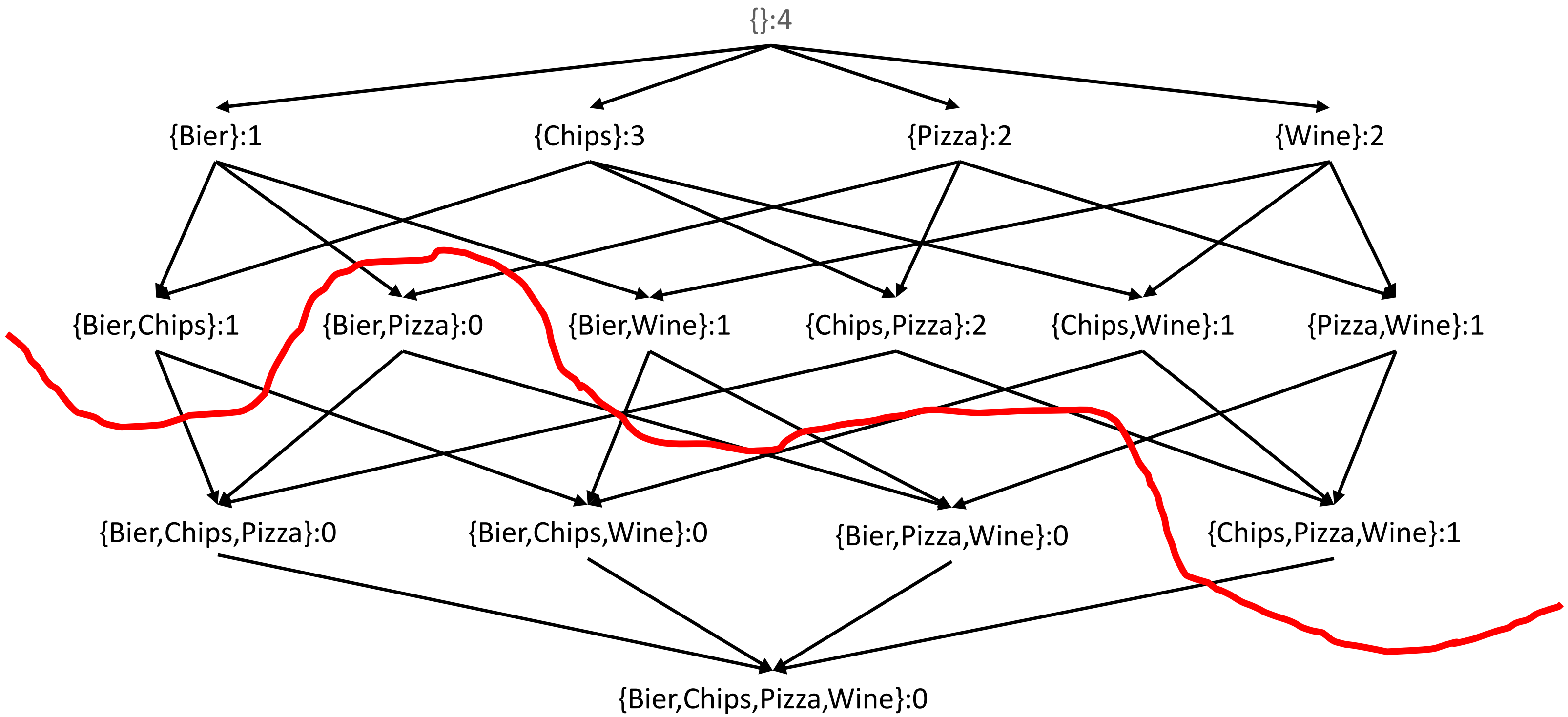
{Chips, Pizza}
{Beer, Chips}
{Chips, Pizza, Wine}
{Wine}

minSupport *s* = 2

Search space and pruning

- Border itemsets *X*: all subsets Y $\subset$ X are frequent, all supersets Z $\supset$ X are not frequent



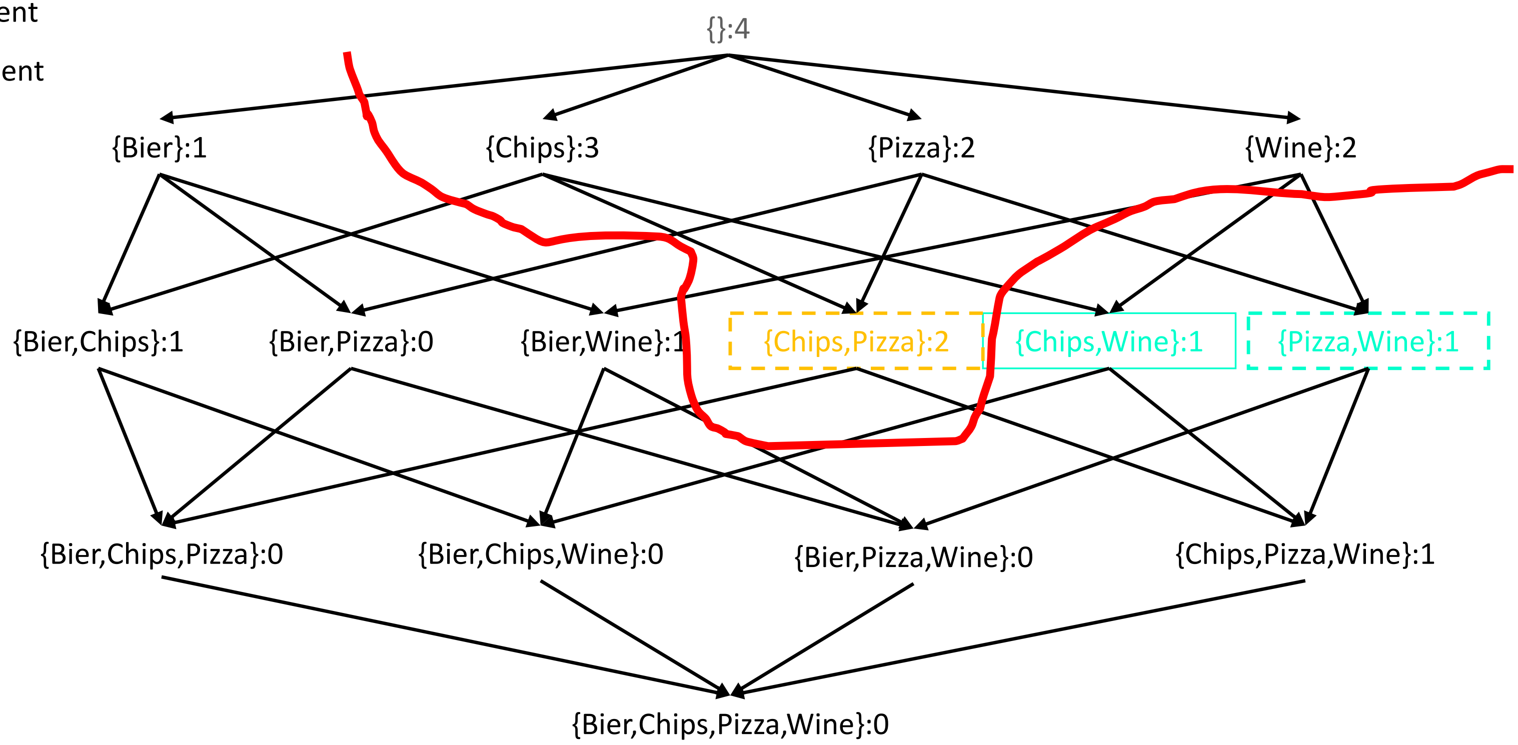


Transaction Database
{Chips, Pizza}
{Beer, Chips}
{Chips, Pizza, Wine}
{Wine}

minSupport *s* = **1**

Search space and pruning

- Border itemsets *X*: all subsets Y ⊂ X are frequent, all supersets Z ⊃ X are not frequent
  - Positive border: *X* is also frequent
  - Negative border: *X* is not frequent



Transaction Database

{Chips, Pizza}
{Beer, Chips}
{Chips, Pizza, Wine}
{Wine}

minSupp = 2

Positive border-itemsets

minSupport *s* = 2

Negative border-itemsets

Frequent itemsets generation: From L_{k-1} to C_k to L_k
L_k: frequent itemsets of size k; C_k: candidate itemsets of size k

A 2-step process:

- Join step: generate candidates C_k
  - L_k is generated by self-joining $C_k = L_{k-1} \bowtie L_{k-1}$, C_k := Set of candidates in L_k
  - Two (k-1)-itemsets p, q are joined, if they agree in the first (k-2) items
- Prune step: prune C_k and return L_k
  - C_k is superset of L_k
  - Naïve idea: count the support for all candidate itemsets in C_k ....$|C_k|$ might be large!
  - Use Apriori property: a candidate k-itemset that has some non-frequent (k-1)-itemset cannot be frequent
    - Prune all those k-itemsets, that have some (k-1)-subset that is not frequent (i.e. does not belong to L_{k-1})
    - Due to the level-wise approach of Apriori, we only need to check (k-1)-subsets
  - For the remaining itemsets in C_k, prune by support count (DB)

Example:
Let L_3={abc, abd, acd, ace, bcd}

- Join step: $C_4=L_3*L_3$
C_4={abc*abd=abcd; acd*ace=acde}

- Prune step (apriori-based):
acde is pruned since cde is not frequent

- Prune step (DB-based):
check abcd support in the DB

Apriori algorithm (pseudo-code)

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**
    C_{k+1} = 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 $\cup_k L_k$;

Candidate generation (self-join, apriori property)

DB scan

subset function

Prune by support count (ask DB)

Subset function:
- For each transaction T in DB, the subset function must check all candidates in the candidate set C_k whether they are part of the transaction T
- Organize candidates C_k in a hash tree

Example

Database TDB

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

$Sup_{min} = 2$

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

Apriori overview

- Advantages:
  - Apriori property
  - Easy implementation (in parallel also)
- Disadvantages:
  - It requires up to $|I|$ database scans
  - It assumes that the DB is in memory
- Complexity depends on
  - minSupport threshold
  - Number of items (dimensionality)
  - Number of transactions
  - Average transaction length

Outline

- Introduction
- Basic concepts
- Frequent Itemsets Mining (FIM) – Apriori
- Association Rules Mining