Pincer-Search: An Efficient Algorithm for Discovering the Maximum Frequent Set

Dao-I Lin Zvi M. Kedem

TKDE 2002 (pp: 553-566)

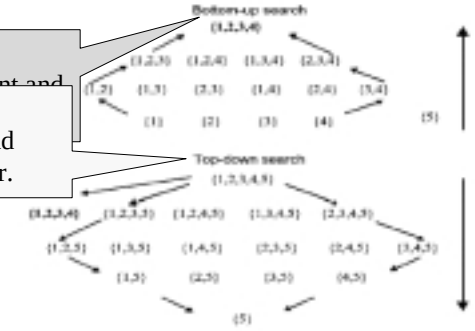
Presented by: Yi-Hung Wu
Date: 2002/9/18

Problem

- **Finding the Maximum Frequent Sets (MFS)**
  - MFS could be long
  - Two-way search: bottom-up and top-down
    - Property 1

If an itemset is infrequent,
all its supersets must be infrequent and
they need not be examined further.

If an itemset is frequent,
all its subsets must be frequent and
they need not be examined further.



P.2

Basics

- **Finding Frequent Sets**
  - Apriori algorithm
    - Join procedure
    - Prune procedure

Algorithm: Apriori algorithm
Input: a database and a user-defined minimum support
Output: all frequent itemsets

```
1. L_0 := ∅; k := 1;
2. C_1 := {{i} | i ∈ I}
3. Answer := ∅
4. while C_k ≠ ∅
5.   read database and count supports for C_k
6.   L_k := {frequent itemsets in C_k}
7.   C_{k+1} := Apriori-gen(L_k)
8.   k := k + 1
9.   Answer := Answer ∪ L_k
10. return Answer
```

Algorithm: The *join* procedure of the Apriori-gen algorithm
Input: L_k, the set containing frequent itemsets found in pass k
Output: preliminary candidate set C_{k+1}

```
/* The itemsets in L_k are sorted */
1. for i from 1 to |L_k − 1|
2.   for j from i + 1 to |L_k|
3.     if L_k.item_1 and L_k.item_j have the same (k − 1)-prefix
4.       C_{k+1} := C_{k+1} ∪ {L_k.item_1 ∪ L_k.item_j}
5.     else
6.       break
```

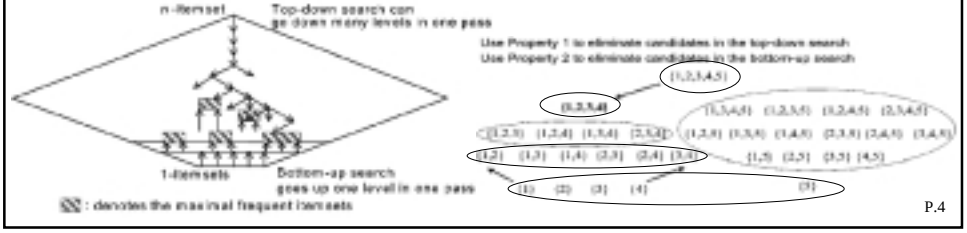
Algorithm: The *prune* procedure of the Apriori-gen algorithm
Input: Preliminary candidate set C_{k+1} generated from the *join* procedure above
Output: final candidate set C_{k+1} which does not contain any infrequent subset

```
1. for all itemsets c in C_{k+1}
2.   for all k-subsets s of c
3.     if s ∉ L_k
4.       delete c from C_{k+1}
```

P.3

Solution (1/3)

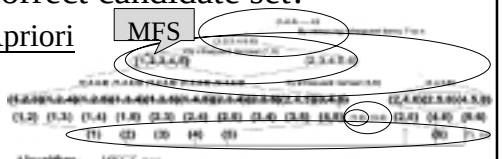
- **Main Idea**
  - Use of the information gathered in one direction to prune more candidates/passes in the other direction
  - Two way search by using the MFCS
    - All the maximal sets that are not known to be infrequent



P.4

Solutions (2/3)

- **Main Issues**
  - How to update the MFCS efficiently?
    - MFCS-gen
  - How to generate the correct candidate set?
    - Join procedure as in Apriori
    - Recovery procedure
    - New prune procedure



Algorithm: The *recovery* procedure
Input: C_{k+1} from *join* procedure, L_k, and current MFS
Output: a complete candidate set C_{k+1}
1. **for** all itemsets ℓ in L_k
2. **for** all itemsets m in MFS
3. **if** the first $k-1$ items in ℓ are also in m
4. /* suppose $\ell.item_{k-1} < l.item_{k-1}$ */
5. **for** i **from** $j+1$ **to** $|m|$
6. $C_{k+1} := C_{k+1} \cup \{\{\ell.item_1, \ell.item_2, ..., \ell.item_k, m.item_i\}\}$

Algorithm: *MFCS-gen*
Input: old MFCS and the infrequent set S_i found in pass i
Output: new MFCS
1. **for** all itemsets $s \in S_i$
2. **for** all itemsets $m \in$ MFCS
3. **if** s is a subset of m
4. MFCS := MFCS $\setminus \{m\}$
5. **for** all items $e \in$ itemset s
6. **if** $m \setminus \{e\}$ is not a subset of any itemset in the MFCS
7. MFCS := MFCS $\cup \{m \setminus \{e\}\}$
8. **return** MFCS

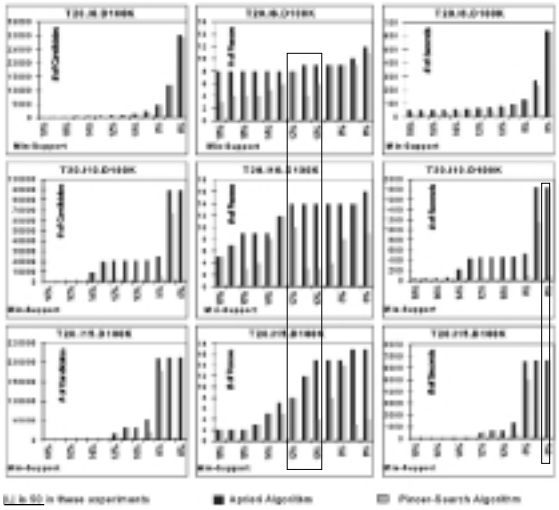
Solutions (3/3)

- **The Basic Pincer-Search Algorithm**
- **The Adaptive Pincer-Search Algorithms**
  - Delay the maintenance of the MFCS
  - Generate the candidates in MFS without counting

Algorithm: The Pincer-Search algorithm
Input: a database and a user-defined minimum support
Output: MFS which contains all maximal frequent itemsets
1. $L_0 := \emptyset$; $k := 1$; $C_1 := \{\{i\} \mid i \in I\}$
2. MFCS := $\{\{1,2,\ldots,n\}\}$; MFS := $\emptyset$
3. **while** $C_k \neq \emptyset$
4. read database and count supports for C_k and MFCS
5. remove frequent itemsets from MFCS and add them to MFS
6. L_k := {frequent itemsets in C_k} $\setminus$ {subsets of MFS}
7. S_k := {infrequent itemsets in C_k}
8. call the MFCS-gen algorithm if $S_k \neq \emptyset$
9. call the *join* procedure to generate C_{k+1}
10. **if** any frequent itemset in C_k is removed in lines 6
11. call *recovery* procedure to recover candidates to C_{k+1}
12. call new *prune* procedure to prune candidates in C_{k+1}
13. $k := k+1$
14. **end-while**
15. **return** MFS

Performance Evaluation (1/2)

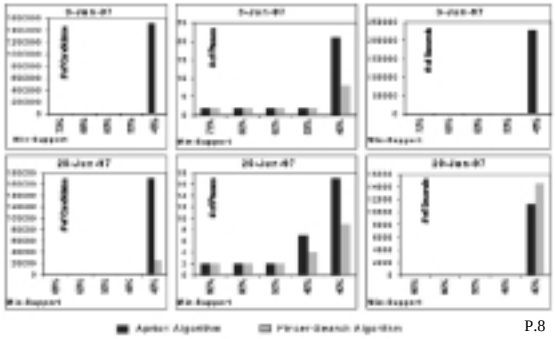
- **Scattered DB**
  - Many short MFS
- **Concentrated DB**
  - Few long MFS



P.7

Performance Evaluation (2/2)

- **Census DB**
  - Very long MFS
  - Scattered distribution
- **Stock Market DB**
  - 3000 stocks
  - Up/down
  - 60 minutes
  - 15 transactions



P.8

Conclusion Remarks

- **Contribution**
  - A new and efficient algorithm for mining maximal frequent sets for applications with long patterns
- **Weakness**
  - Long patterns in a scattered distribution
- **Related Work**
  - Dynamic Item Counting (DIC) [SIGMOD97]
  - Max-Miner [SIGMOD98]

P.9

Paper Scoring

- **Scores {bad, marginal, good, excellent}**
  - Originality: good
  - Technical Depth: good
  - Impact/Practicability: good
  - Readability: good
  - Overall: good

P.10

Related Work

- **Dynamic Itemset Counting and Implication Rules for Market Basket Data**
  - S. Brin, R. Motwani, J. D. Ullman, S. Tsur

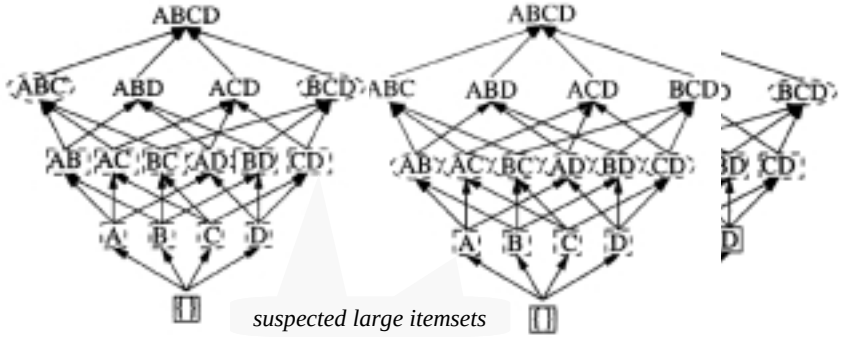
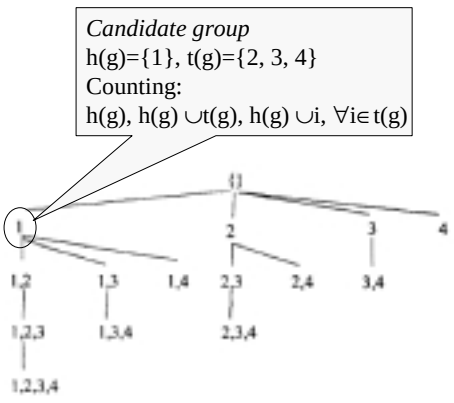


Figure 5: After $2M$ transactions.

Figure 4: After M transactions.

Related Work

- **Efficiently Mining Long Patterns from Databases**
  - R. J. Bayardo



GEN-INITIAL-GROUPS(Data-Set T, Set of Candidate Groups C)
:: C *is passed by reference and returns the candidate groups*
:: *The return value of the function is a frequent 1-itemset*
scan T to obtain F_1, the set of frequent 1-itemsets
impose an ordering on the items in F_1 :: *see section 3.2*
for each item i in F_1 other than the greatest item **do**
  let g be a new candidate with $h(g) = \{i\}$
    and $t(g) = \{j | j$ follows i in the ordering$\}$
  $C \leftarrow C \cup \{g\}$
return the itemset in F_1 containing the greatest item

GEN-SUB-NODES(Candidate Group g, Set of Cand. Groups C)
:: C *is passed by reference and returns the sub-nodes of* g
:: *The return value of the function is a frequent itemset*
remove any item i from $t(g)$ if $h(g) \cup \{i\}$ is infrequent
reorder the items in $t(g)$ :: *see section 3.2*
for each $i \in t(g)$ other than the greatest **do**
  let g' be a new candidate with $h(g') = h(g) \cup \{i\}$
    and $t(g') = \{j | j \in t(g)$ and j follows i in $t(g)\}$
  $C \leftarrow C \cup \{g'\}$
return $h(g) \cup \{m\}$ where m is the greatest item in $t(g)$,
  or $h(g)$ if $t(g)$ is empty.