Efficiently Mining Long Patterns from Databases

Paper presentation by

Dean Cheng

Outline

- Brief Introduction to Max-Miner
- Techniques used in Max-Miner
  - Candidate Itemset Counting
  - Superset Frequency Pruning
  - Item Ordering Policies
  - Subset Infrequency Pruning
  - Support Lower Bounding
- Experiment and Evaluation
- Summary

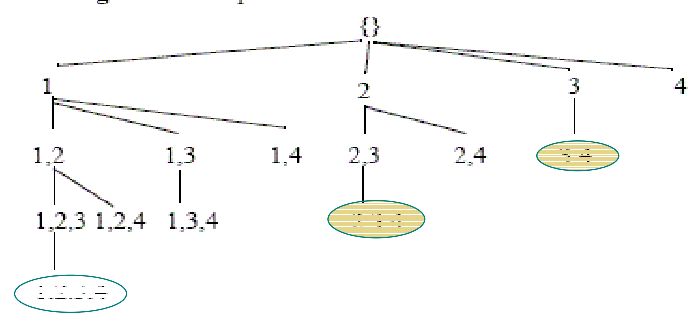
Basic ideas

- An itemset is maximal frequent if it has no superset that is frequent.
- Same as Apriori: scan database, get frequent itemset, get candidate itemset, repeat until no more candidate itemset.
- Look ahead and prune as much as possible.

Max-Miner(1)

- Set-enumeration tree search (breadth-first), utilizing specific ordering and pruning

Figure 1. Complete set-enumeration tree over four items.



Max-Miner(2)

- Each candidate set, g, has two parts: h(g) and t(g). H(g) is the node itself and t(g) is all possible items in the sub-nodes. E.g. h(g) = {1} and t(g) = {2, 3, 4}.

- Counting support of g = counting support of h(g), h(g)∪t(g), and h(g)∪{i} for all i ∈ t(g).

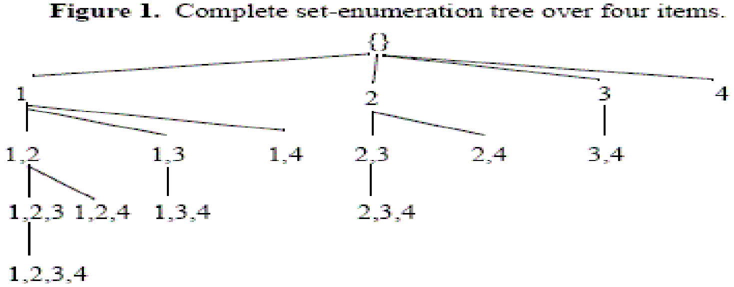
Max-Miner: Pruning(1)

- Superset-frequency pruning: If h(g)∪t(g) is frequent, then all its subsets are frequent but not maximal. Therefore they can be pruned.

- Itemset ordering: order them from least to most frequency. The most frequent items appear in the most candidate groups.

Max-Miner: Pruning(2)

Figure 1. Complete set-enumeration tree over four items.



- Itemset-ordering increase the effectiveness of superset-frequency pruning.

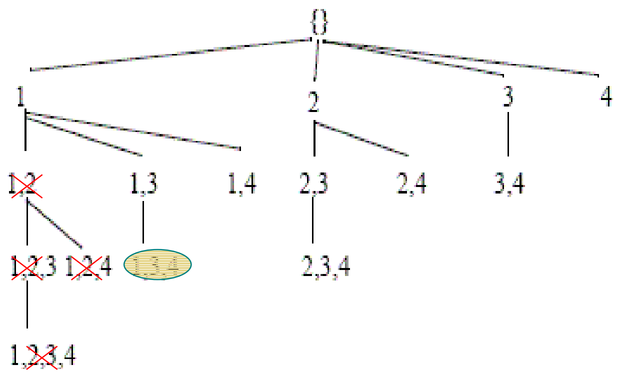
Max-Miner: Pruning(3)

- Subset-infrequency pruning: If h(g)∪{i} is infrequent then all its superset are infrequent. Therefore {i} can be excluded from generating candidate itemset.

- New candidate itemsets are generated from expanding g's sub-nodes.

Example

Figure 1. Complete set-enumeration tree over four items.



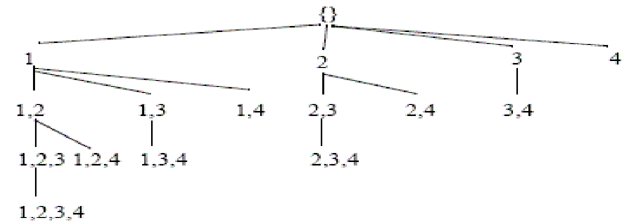
h(g') = {1, 3} t(g') = {4}

Max-Miner Details(1)

Figure 2. Max-Miner at its top level.

MAX-MINER(Data-set T)
;; *Returns the set of maximal frequent itemsets present in* T
Set of Candidate Groups $C \leftarrow \{\ \}$
Set of Itemsets $F \leftarrow \{$GEN-INITIAL-GROUPS$(T, C)\}$
while C is non-empty **do**
scan T to count the support of all candidate groups in C
for each $g \in C$ such that $h(g) \cup t(g)$ is frequent **do**
$F \leftarrow F \cup \{h(g) \cup t(g)\}$
Set of Candidate Groups $C_{new} \leftarrow \{\ \}$
for each $g \in C$ such that $h(g) \cup t(g)$ is infrequent **do**
$F \leftarrow F \cup \{$GEN-SUB-NODES$(g, C_{new})\}$
$C \leftarrow C_{new}$
remove from F any itemset with a proper superset in F
remove from C any group g such that $h(g) \cup t(g)$
has a superset in F
return F

Figure 1. Complete set-enumeration tree over four items.



Max-Miner Details(2)

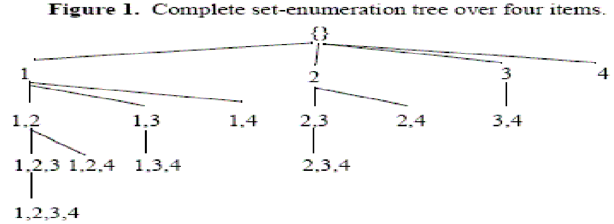
Figure 4. Generating sub-nodes.

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 $t(g') = \{j \mid j \in t(g)$ 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.

- **Order tail items of a group g in increasing order of sup(h(g)∪{i})**

Max-Miner Correctness

Figure 1. Complete set-enumeration tree over four items.



- **The tree enumerates all possible itemsets and Max-Miner will traverse the entire tree unless a node is infrequent or it is a subset of a frequent itemset.**

Support Lower-Bounding(1)

- Compute a lower-bound on the support of an itemset, if lower-bound > minimum support, then all its subset can be pruned.

- drop(I_s, i) = sup(I_s) – sup($I_s \cup$i)

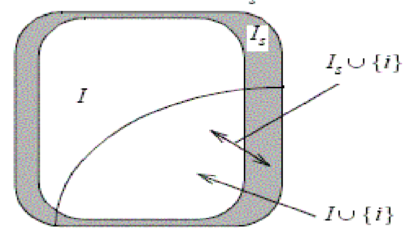
The # of transaction “dropped” when an itemset is extended with an item.

e.g. {1,2,3} {1,2}: sup(1,2) = 1, sup(1,2,3) = 0.5

Support Lower-Bounding(2)

- Theorem (support lower-bounding): sup(I) – drop(I_s, i) is a lower-bound on the support of itemset I$\cup${i} when $I_s \subset$ I.

Figure 5. Illustration of support drop resulting from extending itemsets I and I_s with i.



Support Lower-Bounding(3)

- Theorem (Generalized support lower-bounding): lower-bound on the support of itemset I $\cup$ T where T is an itemset disjoint from I and $I_s \subset$ I

$$\sup(I) - \sum_{i \in T} \mathrm{drop}(I_s, i)$$

I = h(g) **T = t(g)**

Max-Miner: Support LB(1)

- During candidate generation, $h(g_2) \cup t(g_2) \subset h(g_1) \cup t(g_1)$. If $h(g_1) \cup t(g_1)$ is frequent, then no need to expand sub-nodes further.

Figure 6. Generating sub-nodes with support lower-bounding.

```
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) ∪ {i} is infrequent
  reorder the items in t(g)
  for each i ∈ t(g) in increasing item order do
      let g' be a new candidate with h(g') = h(g) ∪ {i}
          and t(g') = {j|(j ∈ t(g)) and j follows i in t(g)}
      if COMPUTE-LB(g', h(g)) ≥ minsup
          then return h(g') ∪ t(g') ;; this itemset is frequent
      else C ← C ∪ {g'}
  return h(g) ;; This case arises only if t(g) is empty
```

Max-Miner: Support LB(2)

Figure 7. Computing the support lower-bound.

```
COMPUTE-LB(Candidate Group g, Itemset I_s)
  ;; Returns a lower-bound on the support of h(g) ∪ t(g)
  ;; Itemset I_s is a proper subset of h(g)
  Integer d ← 0
  for each i ∈ t(g) do
      d ← d + drop(I_s, i)
  return sup(h(g)) – d
```

- **drop(I_s, i) = sup(I_s) – sup($I_s \cup$i)**

Sup(I_s) = sup(old h(g))

Sup($I_s \cup$i) = sup(old h(g)$\cup${i})

Implementation Details

- **Max-Miner uses similar data structure to Apriori.**
- **Max-Miner uses a hash tree to index only the head of each candidate group.**
- **During the second pass, a 2-D array is used and support for h(g)$\cup$t(g) is not counted.**

Experiment and Evaluation(1)

- **Three algorithms: Max-Miner, Apriori, Apriori-LB (a support lower bound version of Apriori). Both Apriori algorithms were optimized for finding maximal frequent itemset.**

Experiment and Evaluation(2)

- **200MHz Power-PC with 256 megabytes. All algorithms were implemented in C++ using same hash tree implementation.**

Table 1. Width and height of the evaluation data-sets.

Data-set	Records	Avg. Record Width
chess	3,196	37
connect-4	67,557	43
mushroom	8,124	23
pumsb	49,046	74
pumsb*	49,046	50
retail	213,972	31
splice	3,174	61

Experiment and Evaluation(3)

Figure 8. CPU time on pumsb*.

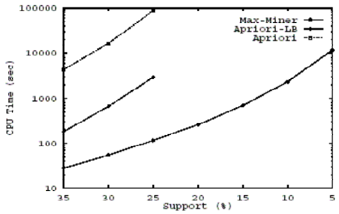


Figure 9. CPU time on mushroom.

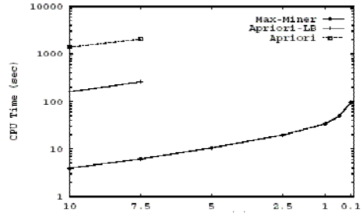


Figure 10. CPU time on chess.

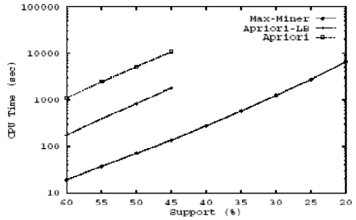
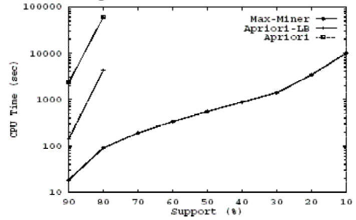


Figure 11. CPU time on connect-4.



Experiment and Evaluation(4)

- Max-Miner is scaling roughly linearly with the number of maximal frequent itemsets.

Figure 14. CPU time per maximal frequent itemset.

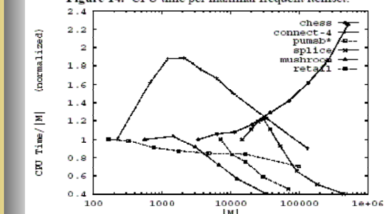
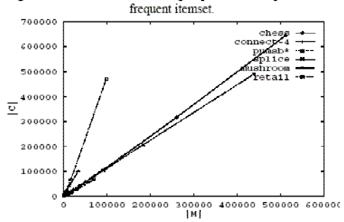


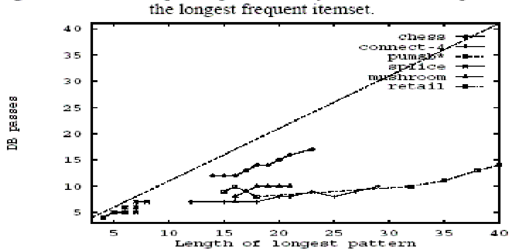
Figure 15. Number of candidate groups considered per maximal frequent itemset.



Experiment and Evaluation(5)

- Number of data passes against the length of the longest patterns identified during each run. Effects of pruning.

Figure 16. Database passes performed by Max-Miner compared to the longest frequent itemset.



Experiment and Evaluation(6)

- More observations:
  - Max-Miner is an order of magnitude faster with the item-ordering heuristic.
  - Performance of Max-Miner without support lower bounding decrease substantially.
  - Support lower-bounding is more effective with datasets that have long patterns.

Summary(1)

- Max-Miner is a new algorithm that applies several new techniques. These techniques can be extended in many way and applied to other algorithms.
- Compare to Apriori, Max-Miner is an efficient algorithm for finding maximal frequent patterns.
- Max-Miner is easily made to incorporate additional constraints during search.

Summary(2)

- One such constrain is finding "longest maximal pattern only" to reduce space and time further (Max-Miner-LO).
- More works can be done adding more constraints to Max-Miner during search.
- More comparisons to other maximal frequent pattern finding algorithms need to be done.

Thank You

Questions?