Data Mining
FP-Tree Approach
(A radically different approach)

Sharma Chakravarthy
Information Technology Laboratory (IT Lab)
Computer Science and Engineering Department
The University of Texas at Arlington, Arlington, TX

Email: sharma@cse.uta.edu
URL: http://itlab.uta.edu/sharma

Compact Representation of Frequent Itemsets

- FP-growth (Frequent Pattern growth)
  - avoids repeated scans of the database (used in Apriori) by using a compressed representation of the transaction database using a data structure called FP-tree
  - Avoids candidate itemset explosion
  - Constructs FP-tree once and uses it in a recursive divide- and-conquer approach to mine the frequent itemsets

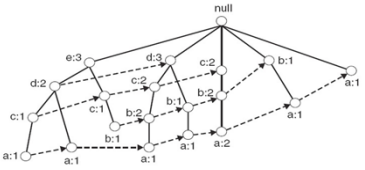


Figure 6.25. An FP-tree representation for the data set shown in Figure 6.24 with a different item ordering scheme.

Compact Representation of Frequent Itemsets

- In practice, the number of frequent itemsets produced from transaction data can be very large, when
  - the database is dense i.e. many items per transaction on average
  - the number of transactions is large
  - the min_sup is set too low
- We will look at methods that
  - use the properties of the itemset lattice and the support function
  - compress the collection of frequent itemsets into a more manageable size
  - Allow us to derive all frequent itemsets from the compressed representation

Compact Representation of Frequent Itemsets

- FP-tree is a compressed representation of the transaction database
- Each transaction is mapped onto a path in the tree
- Each node contains an item and the support count corresponding to the number of transactions with the prefix corresponding to the path from root
- Nodes having the same item label are cross-linked: this helps finding the frequent itemsets ending with a particular item

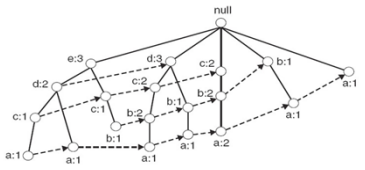


Figure 6.25. An FP-tree representation for the data set shown in Figure 6.24 with a different item ordering scheme.

FP-Tree Construction

TID	Items
1	{A,B}
2	{B,C,D}
3	{A,C,D,E}
4	{A,D,E}
5	{A,B,C}
6	{A,B,C,D}
7	{A}
8	{A,B,C}
9	{A,B,D}
10	{B,C,E}

Horizontal data format

- In the first pass, 1-itemset support is counted and only frequent 1-itemsets (F1) are retained. Infrequent items are discarded.
- items in each transaction are ordered using frequency in descending order (why?)
  - Most frequent items are likely to be in more prefixes
  - It can also be used in ascending order; Tree constructed will be different and depends on the order in which Txs are used to construct the tree
- Lexicographic order of items is not used

Vertical data format is Also possible/used in other algorithms

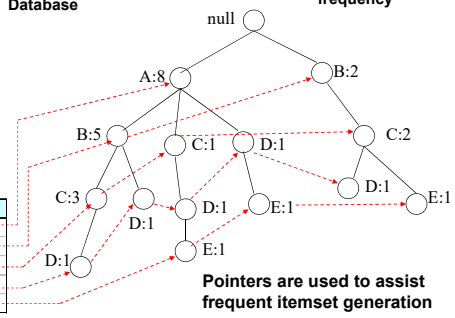
Item	frequency
A	8
B	7
C	6
D	5
E	3

FP-Tree Construction (at the end)

TID	Items
1	{A,B}
2	{B,C,D}
3	{A,C,D,E}
4	{A,D,E}
5	{A,B,C}
6	{A,B,C,D}
7	{B,C}
8	{A,B,C}
9	{A,B,D}
10	{B,C,E}

Transaction Database

Descending frequency



Header table

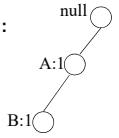
Item	Pointer
A	
B	
C	
D	
E	

Pointers are used to assist frequent itemset generation

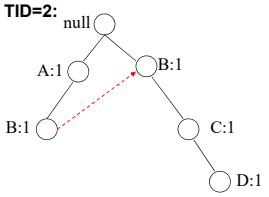
FP-tree construction (second pass)

TID	Items
1	{A,B}
2	{B,C,D}
3	{A,C,D,E}
4	{A,D,E}
5	{A,B,C}
6	{A,B,C,D}
7	{A}
8	{A,B,C}
9	{A,B,D}
10	{B,C,E}

After reading TID=1:



After reading TID=2:



FP-Tree for ascending frequency

- If the transactions share a significant number of items, FP-tree can be considerably smaller as the common subset of the items is likely to share paths
- Notice any difference between this and previous tree?
- More bushy because less paths share prefixes due to ascending order!

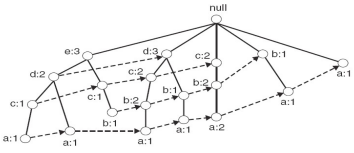


Figure 6.25. An FP-tree representation for the data set shown in Figure 6.24 with a different item ordering scheme.

Observations

- The size of an FP-tree is typically smaller than the size of the uncompressed data because many transactions share a few items in common
- You can think of best and worst case for the tree pattern and size
- The size of the FP-tree also depends on how the items are ordered
- Ascending and descending orders can make a huge difference
- Additional list of items and pointers are used.
- It may not be possible to hold the FP-tree in memory

Mining Frequent Patterns

- How do we get all frequent patterns from the FP-Tree?
- A "divide and conquer" strategy is used
- Explores the tree in a bottom-up manner.
- Intuitively:
  1. Find all frequent patterns containing one of the items
  2. Then find all frequent patterns containing the next item (suffix) but NOT containing the previous one
  3. Repeat 2) until we are out of items
- Start from the bottom of the header table
- Extract path corresponding to e

Mining Frequent Patterns

- To find all frequent itemsets ending with given last item (e.g. e), we first need to compute the support of the item
  - This is given by the sum of support counts of all nodes labeled with the item (σ(E)=3)
  - found by following the cross-links connecting the nodes with the same item
- If last item is found frequent, FP-growth next iteratively looks for all frequent itemsets ending with given item
  - Of length-2 suffix (DE,CE,BE, and AE),
  - and recursively with length-3 suffix, length-4 suffix until no more frequent itemsets are found
- Conditional FP-tree is constructed for each different suffix to speed up the computation

Frequent itemset generation in FP-growth

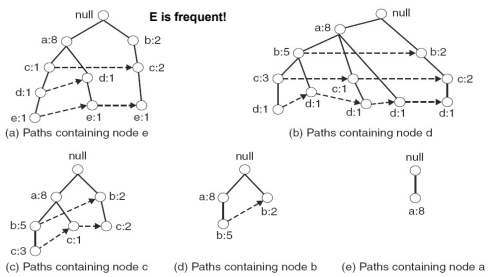


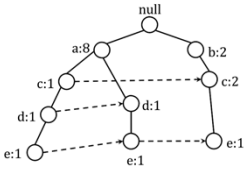
Figure 6.26. Decomposing the frequent itemset generation problem into multiple subproblems, where each subproblem involves finding frequent itemsets ending in e, d, c, b, and a.

Mining Frequent Patterns

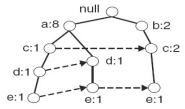
Here min_sup is 2

TID	Items
1	{a, b}
2	{b, c, d}
3	{a, c, d, e}
4	{a, d, e}
5	{a, b, c}
6	{a, b, c, d}
7	{a}
8	{a, b, c}
9	{a, b, d}
10	{b, c, e}

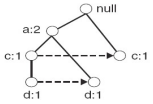
Prefix paths ending in e



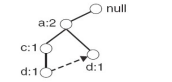
Frequent itemset generation in FP-growth



(a) Prefix paths ending in e

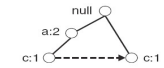


(b) Conditional FP-tree for e



(c) Prefix paths ending in de

(d) Conditional FP-tree for de



(e) Prefix paths ending in ce

(f) Prefix paths ending in ae

Figure 6.27. Example of applying the FP-growth algorithm to find frequent itemsets ending in e.

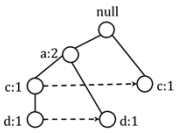
Mining Frequent Patterns

- Because {e} is frequent, the algorithm has to solve the sub problems of finding frequent itemsets ending in e.
- That is, de, ce, be, and ae (where do we get this from?)
- For that, we convert the prefix paths into a **conditional FP-tree**, which is structurally similar to a FP-Tree, except it is used to find frequent itemsets ending with a particular suffix.
- It is done as follows.
- This process is repeated for itemsets ending in d, c, b, and a

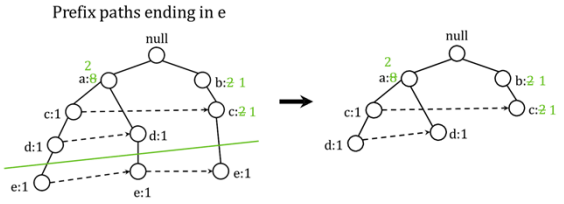
Example of Conditional FP-Tree ending in e

TID	Items
1	{a, b}
2	{b, c, d}
3	{a, c, d, e}
4	{a, d, e}
5	{a, b, c}
6	{a, b, c, d}
7	{a}
8	{a, b, c}
9	{a, b, d}
10	{b, c, e}

Conditional FP-tree for e



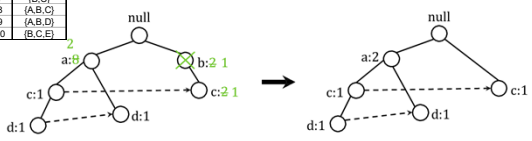
Example of Conditional FP-Tree ending in e



- To obtain the conditional FP-Tree for e from the prefix sub-tree ending in e, remove the nodes containing e (as this information is no longer needed)

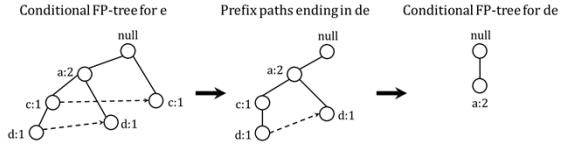
Example of Conditional FP-Tree ending in e

TID	Items
1	{A,B}
2	{B,C,D}
3	{A,C,D,E}
4	{A,D,E}
5	{A,B,C}
6	{A,B,C,D}
7	{B,C}
8	{A,B,C}
9	{A,B,D}
10	{B,C,E}



- Now remove infrequent items (nodes) from the prefix paths. In this example, b has a support of 1 (note this really means b has a support of 1).
- That is, there is only 1 tx containing b and e

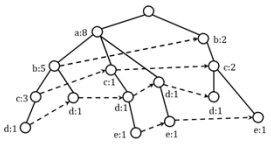
Example of Conditional FP-Tree ending in e



- Use the conditional FP-tree for e to find frequent itemsets ending in de, ce, and ae (be is not considered as b is not in the conditional FP-tree for e)
- For each item, find the prefix paths from conditional tree for e, generate the conditional FT-tree, and continue recursively

Example of Conditional FP-Tree ending in e

Given the example tree below, FP-growth looks for itemsets ending in e first, followed by, d, b, and finally a



Suffix	Frequent Itemsets
e	{e}, {d, e}, {a, d, e}, {c, e}, {a, e}
d	{d}, {c, d}, {b, c, d}, {a, c, d}, {b, d}, {a, b, d}, {a, d}
c	{c}, {b, c}, {a, b, c}, {a, c}
b	{b}, {a, b}
a	{a}

- Since every transaction is mapped onto a path in the FP-tree, we can derive the frequent itemsets ending with a particular item by examining only the paths containing the item's nodes

Summary

- A divide and conquer algorithm is used to generate frequent itemsets
- At each recursive step. A conditional FP-tree is constructed by updating the frequency counts along the prefix paths and removing all infrequent items
- Because the subproblems are disjoint, FP-growth will not generate any duplicates
- Uses a compact representation and efficient generation of frequent itemsets
- For certain transaction data sets, FP-growth outperforms the standard Apriori algorithm by several orders of magnitude

Thank You !!!

For more information visit:
http://itlab.uta.edu

Discussion