

Mining Frequent Patterns without Candidate Generation  
Jiawei Han, Jian Pei and Yiwen Yin  
Presentation Summary

Most of the mining methods for generation of the frequent patterns use apriori method.

There are two disadvantage of Apriori-like method:

- (1) It may need to generate a huge number of candidate sets. For example, if there are  $10^4$  frequent 1-itemsets, the Apriori algorithm will need to generate more than  $10^7$  candidate 2-itemsets and accumulate and test their occurrence frequencies.
- (2) It may need to repeatedly scan the database and check a large set of candidates by pattern matching.

Therefore, the author of paper came up with a solution:

- Constructed a data structure called FP-Tree (Frequent Pattern Tree) from database.
- Mining frequent patterns using FP-Tree.

FP-Tree:

It consists of one root labeled as null, a set of item prefix subtrees as the children of the root, and a frequent –item header table.

Each node in the item prefix subtree consists of three fields: item-name, count and node link where--- item-name registers which item the node represents, count registers the number of transactions represented by the portion of path reaching this node, node link links to the next node in the FP- tree.

Each item in the header table consists of two fields---item name and head of node link, which points to the first node in the FP-tree carrying the item name.

Algorithm 1(FP-tree construction)

- 1) Scan the transaction database DB once. Collect the set of frequent items F and their supports. Sort F in support descending order as L, the list of frequent items.

*Frequency Count of items (by id):*

| Item ID | frequency count |
|---------|-----------------|
| 2       | 6               |
| 4       | 5               |
| 16      | 4               |
| 10      | 4               |
| 35      | 4               |
| 14      | 3               |
| 18      | 3               |
| 37      | 3               |
| 43      | 3               |
| 7       | 3               |

- 2) Create the root of an FP-tree, T, and label it as null. For each transaction in the database do the following.

-Select and sort the frequent item in each transaction according to the order of L  
Let the sorted frequent item list be [p|P], where p is the first element and P is the remaining list. Call insert for each item insert ([p|P], T).

-insert function.

```
insert([p|P],T)
{
// check if T has a child N where N.item-name=p.item-name then increment N count by
1.
//else create a new node with count 1,its parent linked to T,and its node-link be linked to
nodes with the same item-name via node-link structure
//call insert till P is non empty.
}
```

Example:

*Transaction Example (from assignment #2, we picked 10 transactions with minimum support of 3)*

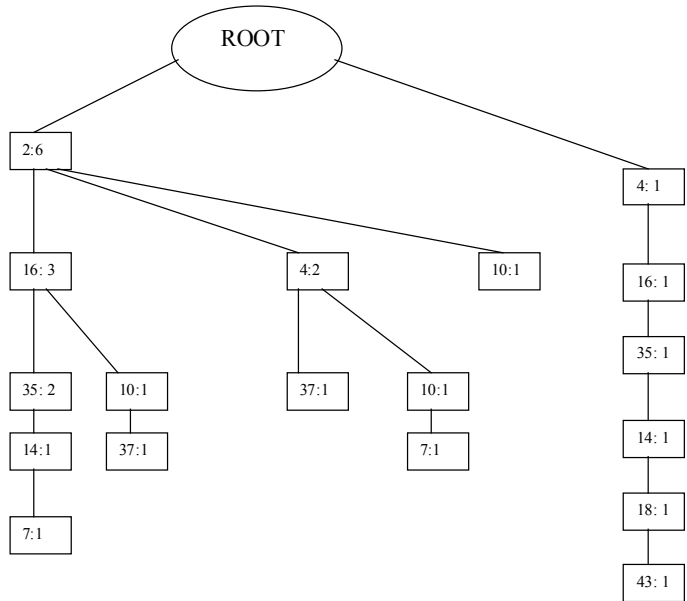
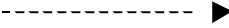
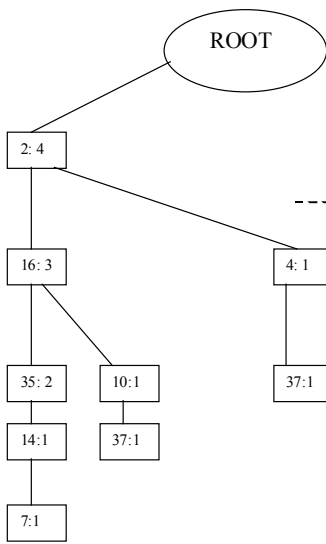
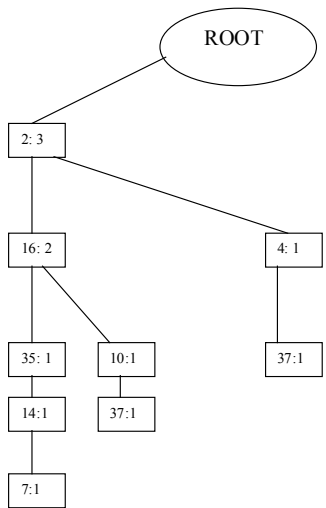
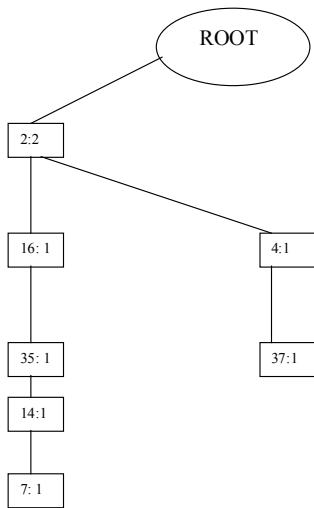
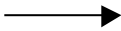
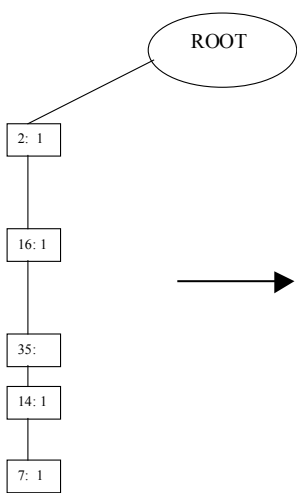
| TID  | Items in Basket  | (Ordered)<br>Frequent Items |
|------|--|-----------------------------|
| 100  | <b>2</b> 7 11 13 <b>14</b> <b>16</b> 31 <b>35</b> 36               | 2 16 35 14 7                |
| 200  | <b>2</b> <b>4</b> 20 23 28 <b>37</b> 44 56 60                      | 2 4 37                      |
| 300  | <b>2</b> <b>10</b> <b>16</b> 20 23 26 33 <b>37</b> 72              | 2 16 10 37                  |
| 400  | <b>2</b> <b>16</b> 24 26 27 30 33 <b>35</b> 51                     | 2 16 35                     |
| 500  | <b>2</b> <b>4</b> <b>7</b> 9 <b>10</b> 17 51 56 87                 | 2 4 10 7                    |
| 600  | <b>2</b> <b>10</b> 11 21 24 27 29 40 45                            | 2 10                        |
| 700  | 3 <b>4</b> 11 <b>14</b> <b>16</b> <b>18</b> 29 <b>35</b> <b>43</b> | 4 16 35 14 18 43            |
| 800  | 3 <b>7</b> <b>18</b> 19 21 25 26 32 36                             | 18 7                        |
| 900  | <b>4</b> 13 <b>14</b> <b>18</b> <b>37</b> 40 <b>43</b> 50 67       | 4 14 18 37 43               |
| 1000 | <b>4</b> <b>10</b> 31 32 <b>35</b> <b>43</b> 45 51 65              | 4 10 35 43                  |

*Frequency Count of items (by id):*

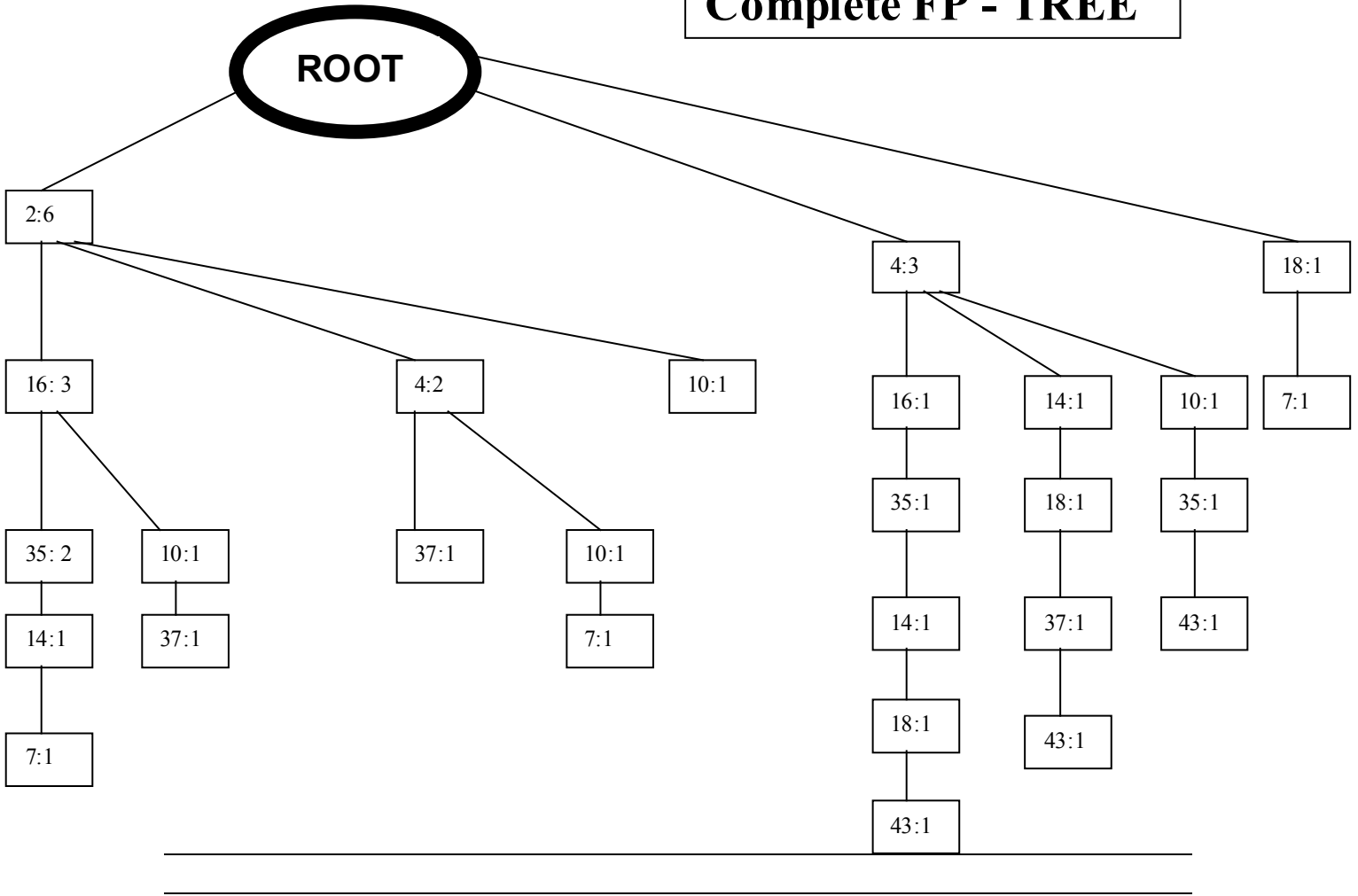
| Item ID | frequency count |
|---------|-----------------|
| 2       | 6               |
| 4       | 5               |
| 16      | 4               |
| 10      | 4               |
| 35      | 4               |
| 14      | 3               |
| 18      | 3               |
| 37      | 3               |
| 43      | 3               |
| 7       | 3               |

Following is the complete FP-Tree based on the previous data.

---



# Complete FP - TREE



## Mining Frequent Patterns using FP-growth:

Based on the FP-Tree characteristic, we can develop an efficient mining method for mining the complete set of frequent patterns. FP-growth method transforms the problem of finding long frequent pattern to looking for shorter ones recursively and then concatenating the suffix. It uses the least frequent items as a suffix, offering good selectivity.

The mining of the FP-tree proceeds as follows. Start from each frequent length-1 pattern; construct its conditional pattern base (a sub database which consists of the set of prefix paths in the FP-tree co-occurring with the suffix pattern), then construct its FP-tree and perform mining recursively on such a tree.

### Algorithm 2

Input: constructed FP-tree

Output: complete set of frequent patterns

Method: Call FP-growth (FP-tree, null).

procedure FP-growth (Tree,  $\alpha$ )

procedure FP-growth (Tree,  $\alpha$ )

{

if Tree contains a single path P

then for each combination do generate pattern  $\beta \cup \alpha$  with support = minimum support of nodes in  $\beta$ .

Else

For each header  $a_i$  in the header of Tree do {

    Generate pattern  $\beta = a_i \cup \alpha$  with

    support =  $a_i$ .support;

    Construct  $\beta$ 's conditional pattern base and

    then  $\beta$ 's conditional FP-tree Tree  $\beta$

    If Tree?  $\neq$  null

    Then call FP-growth (Tree  $\beta$ ,  $\beta$ )

}

| ITEM | Conditional patterns base  | Conditional FP-Tree  | frequent patterns generated  |
|------|--|--|--|
| 7    | {(2:1, 16:1, 35:1, 14:1),<br>(2:1, 4:1, 10:1), (18:1)}                           | {(2: 2)}   7   | 2 7 : 2  |
| 43   | {(4:1,16:1, 35:1, 14:1, 18:1),<br>(4:1, 14:1, 18:1, 37:1),<br>(4:1, 10:1, 35:1)} | {(4:1, 35:1, 14:1, 18:1),<br>(4:1, 14:1, 18:1),<br>(4:1, 35:1)}   43 | 4 43:2, 35 43:2, 14 43 :2, 18 43:2,<br>4 35 43:2, 4 14 43:2, 4 18 43:2, 14 18 43:2,<br>4 14 18 43 :2 |
| 37   | {2:1, 16:1, 10:1}, {2:1, 4:1},<br>{4:1, 14:1, 18:1}                              | {(2: 2, 4:1), (4:1)}   37  | 2 37:2, 4 37:2   |
| 18   | {4:1, 16:1, 35:1, 14:1}, {4:1,<br>14:1}  | {(4:2, 14:2)}   18   | 4 18:2, 14 18:2, 4 14 18:2   |
| 14   | {2:1, 16:1, 15:1}, {4:1, 16:1,<br>35:1}, {4:1}                                   | {(4: 2)}   14  | 4 14:2   |
| 35   | {2:2, 16:2}, {4:1, 16:1}, {4:1,  | {(2:2, 16:2), (4:2)}   35  | 2 35:2, 16 35:2, 4 35:2, 2 16 35:2   |

|    |                                |                            |                |
|----|--------------------------------|----------------------------|----------------|
|    | 10:1}                          |                            |                |
| 10 | {2:1, 16:1}, {2:1, 4:1}, {4:1} | { (2:2, 4:1), (4:1) }   10 | 2 10:2, 4 10:2 |
| 16 | {2:3}, {4:1}                   | { (2:3) }   16             | 2 16:3         |
| 4  | {2:2}                          | {2:2}   4                  | 4, 2 : 2       |
| 2  | 0                              | 0                          | -----          |

Details of FP-tree mining process: -

### Step 1) Generation of patterns

Let's consider item 7, which is the last item in the frequent item header table. 7 occurs thrice in the complete FP-tree shown. Therefore the three branches are

(2:6, 16:3, 35:2, 14:1,7:1)  
(2:6, 4:2, 10:1, 7:1)  
(18:1, 7:1)

### Step 2) Generation of Conditional Pattern Base

To study which items occur together it is necessary to consider only 7's prefix path. Therefore it is sufficient if we consider only the prefix paths. Also we can see that each branch occurs only once with 7(as shown by 7:1). Reduce the prefix count to that of 7 in each branch.

(2:1, 16:1, 35:1, 14:1)  
(2:1, 4:1, 10:1)  
(18:1)

### Step 3) Generation of Conditional FP-Tree

These prefix paths are called conditional pattern base (sub pattern base under 7's existence). Now we have to construct a conditional FP-tree for 7. The conditional FP-tree consists only of (2:2). 16,35,14,4,10,18 are not included since they occur only once and our support is 2.

The frequent pattern can now be (2,7:2).

Now let's consider 43. This is an interesting case and if understood convinces one that he has understood the algorithm.

43 occurs thrice. (Step 1 not shown) same as above.

Step 2) Consider only the prefix paths of 43 and reduce the prefix count of the all the branches to that of 43. So the conditional pattern base is as follows

(4:1,16:1, 35:1, 14:1, 18:1),  
(4:1, 14:1, 18:1, 37:1),  
(4:1, 10:1, 35:1)

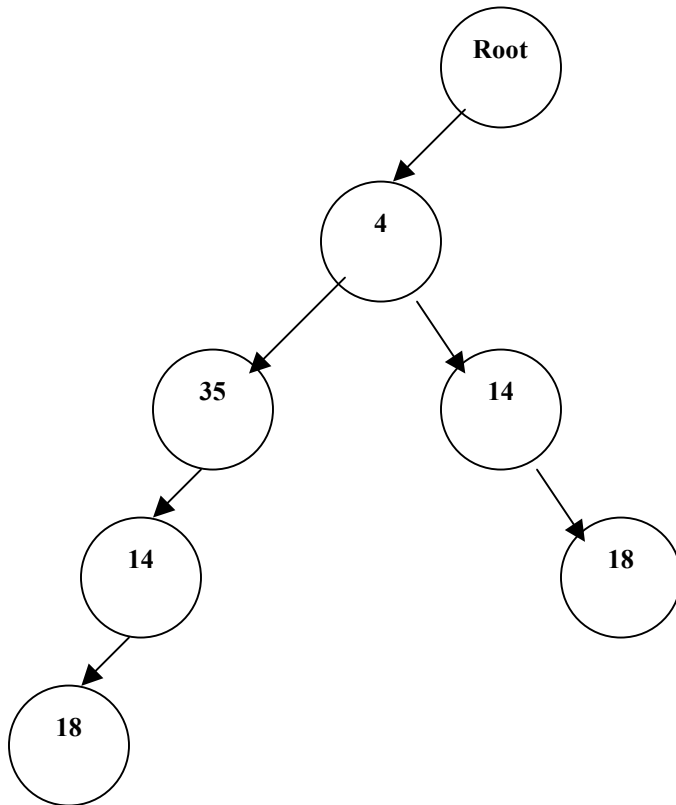
Step3) Construction of conditional FP-tree for 43.

The conditional pattern base reduces to

(4:1, 35:1, 14:1, 18:1)  
(4:1, 14:1, 18:1)

(4:1, 35:1) }

16,37,10 are deleted because they have a count less than 2 in 43's subdatabase. The possible tree that can be generated out of this is –



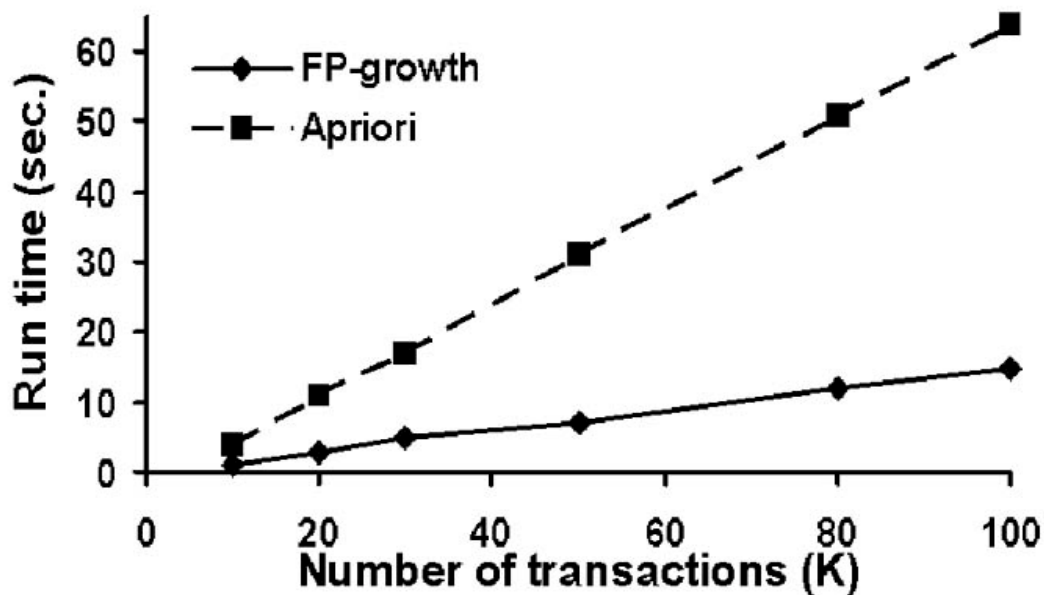
Header Table

| Item | support |
|------|---------|
| 4    | 2       |
| 35   | 2       |
| 14   | 2       |
| 18   | 2       |

This tree is mined recursively to get the frequent patterns as show in the figure.

Opinion:

- Compare to Apriori-like algorithm, it is difficult to implement FP-Tree on actual coding because of the tree structure.
- In case of large database, it is a good candidate to use for short and long patterns because FP-growth scales much better than Apriori. It becomes very obvious when the “support threshold” goes down.
- FP-tree is constructed the way that the higher frequent items are closer to the root (upper portion). Therefore, from a “searching” or “scanning” point of view, it is very easy to select (mining) the items with high threshold by dropping the lower portion of the tree.
- The author has clearly their contribution. At the end of paper, it shown a set of comparisons between Apriori-like method and FP-Tree. For example:



- It clearly shown the superior of FP-Tree. I'm convinced it was a big contribution.
- I think the example (data) of the paper doesn't showed the power of this algorithm. Also, it would help the new user (of the method) if the paper can provide a more complex example. For example, what would happen if a item branch out to a lot of leave node?

### Description of Approach:

- First we read through the paper and try to understand this new algorithm. By doing that, we have to understand what kind of problem does Apriori-like approach has. In other words, we need to understand the weakness of Apriori-like method such that we can extract the advantage of this new approach.
- Then, we went through the example from the paper and see the power of this method.
- We tried pick some sample data which structure is completely different from the paper's example. By doing that, we understand more about this new method and found out how the algorithm really work.
- Problem happens when we picked the sample data. For example, some data we really couldn't see the power of this algorithm (e.g. data size too small, data pattern didn't have enough repeated suffix... etc)
- Also, the algorithm itself is difficult to explain without using any graph. That probably the weakness this algorithm because it is a require a lot of intelligent thinking (create tree structure) to the data set. In other word, there are a lot overhead processes need before creating the tree.