CS 6604: Data Mining Fall 2007

Lecture 5 — Monday, September 3, 2007

Lecture: Naren Ramakrishnan Scribe: Andrew Hall

1 Overview

In the last lecture we developed a basic theory and approach for the mining of rules from itemsets.

In this lecture we explore algorithms that mine without candidate generation. Recall that there are two processes typically involved in data mining of frequent itemsets:

1. Candidate Generation
2. Evaluation

Candidate generation is by far the most time consuming process, so it is desirable to speed this up. This work was the PhD dissertation of Jian Pei, who began publishing in 2000 and already has an h-index rating of about 30. (h-index = C | C papers have C citations or more each).

2 FP-Trees

Jian Pei's algorithm directly mines frequent itemsets without generating candidates. The claim is that by gathering sufficient statistics into a suitable data structure (called an FP tree), all of the frequent patterns can be generated without going back to the database. Only two passes through the database are required to generate the FP Tree, and from the FP Tree, all frequent patterns can be generated.

2.1 Example from paper

tx	Items
1	f,a,c,d,g,i,m,p
2	a,b,c,f,l,m,o
3	b,f,h,j,o
4	b,c,k,s,p
5	a,f,c,e,l,p,m,n

Assume that the minimum support threshold is 3. The algorithm makes the first pass through the database and finds singleton itemsets (items) with enough support:

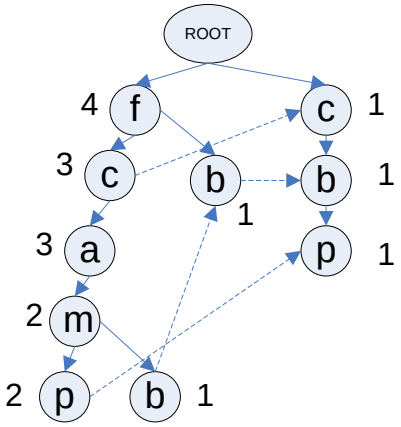

Figure 1: FP-Tree

f: 4	~~i: 1~~
a: 3	~~j: 1~~
b: 3	~~k: 1~~
c: 4	~~l: 2~~
~~d: 1~~	m: 3
~~e: 1~~	~~n: 1~~
~~g: 1~~	~~o: 2~~
~~h: 1~~	p: 3

The algorithm then ranks the frequent items in descending order:

f: 4
c: 4
a: 3
b: 3
m: 3
p: 3

We then go through each transaction at a time, figure out which of the single items are contained in that transation, and insert these items into the FP-Tree data structure based on the prefix path. Once the prefix path deviates from the tree, a new branch is created. The hope is that the FP-Tree won't have to keep adding nodes and edges, just increment the counts at existing nodes as shown in Figure 1.

2.2 FP-Tree Construction: Formal Algorithm

1. Scan the transaction database DB once. Collect F, the set of frequent items, and the support of each frequent item. Sort F in support-descending order as FList, the list of frequent items.

2. Create the root of an FP-tree, T , and label it as null. For each transaction Trans in DB do the following. Select the frequent items in *Trans* and sort them according to the order of FList. Let the sorted

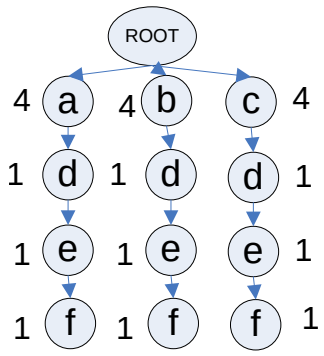

Figure 2: FP-Tree example that is not necessarily compact.

frequent-item list in *Trans* be [p | P], where p is the first element and P is the remaining list. Call *insert_tree*([p | P], T). The function *insert_tree*([p | P], T) is performed as follows. If T has a child N such that *N.item-name* = *p.item-name*, then increment Ns count by 1; else create a new node N, with its count initialized to 1, its parent link linked to T , and its node-link linked to the nodes with the same *item-name* via the node-link structure. If P is nonempty, call *insert_tree*(P, N) recursively.

Notice how different occurrences of a same item are linked up in the data structure. We also have a reference table, not shown here, again sorted in decreasing order of item frequency, that has pointers to the first occurrence of the item in the tree.

FP-trees are intended to be compact, but the tree constructed by the above algorithm need not be compact. Take the tree in Figure 2, which is generated from the below database.

tx	Items
1	a,d,e,f
2	a
3	a
4	a
5	c,d,e,f
6	b
7	b
8	b
9	c,d,e,f
10	c
11	c
12	c

However, a shorter tree will have $d \rightarrow e \rightarrow f$ as the prefix path from the root.

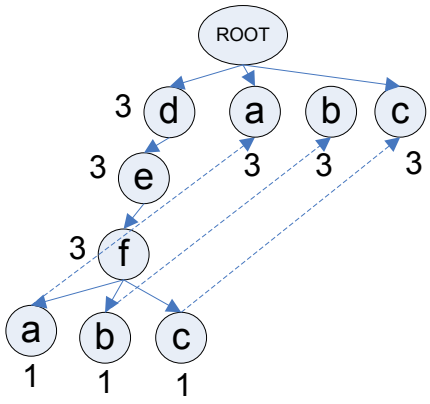

Figure 3: optimal FP-Tree

3 Mining frequent patterns using FP-tree

To find the frequent patterns we start with the reference table (from the bottom) and follow the pointers from node to node .

3.1 Example

As an example we'll generate the frequent patterns from the FP-tree shown in Figure 1.

Node p: its single item frequent pattern is (p:3), and p is contained in two paths in the database (f :4, c:3, a:3,m:2, p:2) and (c:1, b:1, p:1). Notice that the path does contain the string (f, c, a) three times and (f) four times, but they only appear with p twice. To study which string appears with p, only p's prefix path (f :2, c:2, a:2,m:2) (or $\langle fcam : 2\rangle$) is used. The second path indicates string (c, b, p) appears once in the set of transactions, so p's prefix path is $\langle cb : 1\rangle$. These two prefix paths of p, (f cam:2), (cb:1), form p's *subpattern-base*, which is called p's conditional pattern base. Construction of an FP-tree from the conditional pattern-base (called p's conditional FP-tree) leads to one branch (c:3). Which yields only one frequent pattern (cp:3). We are now done with node p.
Node m: its single item frequent pattern is (m:3), and it has two paths, $\langle f : 4, c : 3, a : 3, m : 2\rangle$ and $\langle f : 4, c : 3, a : 3, b : 1, m : 1\rangle$. Even though p appears with m, there is no need to include p here since all frequent patterns involving p were generated while processing node p. Here m's conditional pattern-base is (fca:2), (fcab:1). Constructing the FP-tree, we arrive at m's conditional FP-tree, $\langle f : 3, c : 3, a : 3\rangle$, a single frequent pattern path, as shown in figure 4. We mine the conditional FP-tree recursively by calling mine ($\langle f : 3, c : 3, a : 3\rangle \|$ m). figure 4 shows that mine($\langle f : 3, c : 3, a : 3\rangle \|$ m) involves mining three items (a),(c),(f) sequentially. The first derives a frequent pattern (am:3), a conditional pattern-base (fc:3), and then calling mine($\langle f : 3, c : 3\rangle \|$ am); the second derives a frequent pattern (cm:3), then conditional pattern-base (f:3), and then call mine($\langle f : 3\rangle \|$ cm); and the third derives only a frequent pattern (fm:3). The call mine($\langle f : 3, c : 3\rangle \|$ am) gives two patterns (cam:3) and (fam:3), and the conditional pattern-base (f:3), which results in the call mine($\langle f : 3\rangle \|$ cam), that produces the pattern (fcam:3). Calling mine($\langle f : 3\rangle \|$ cm) produces one pattern (fcm:3). Therefore, the set of frequent patterns produced by m is (m:3), (am:3), (cm:3), (fm:3), (cam:3), (fam:3), (fcam:3), (fcm:3).
Node b: has support (b:3) and has three paths: $\langle f : 4, c : 3, a : 3, b : 1\rangle$, $\langle f : 4, b : 1\rangle$, and $\langle c : 1, b : 1\rangle$.

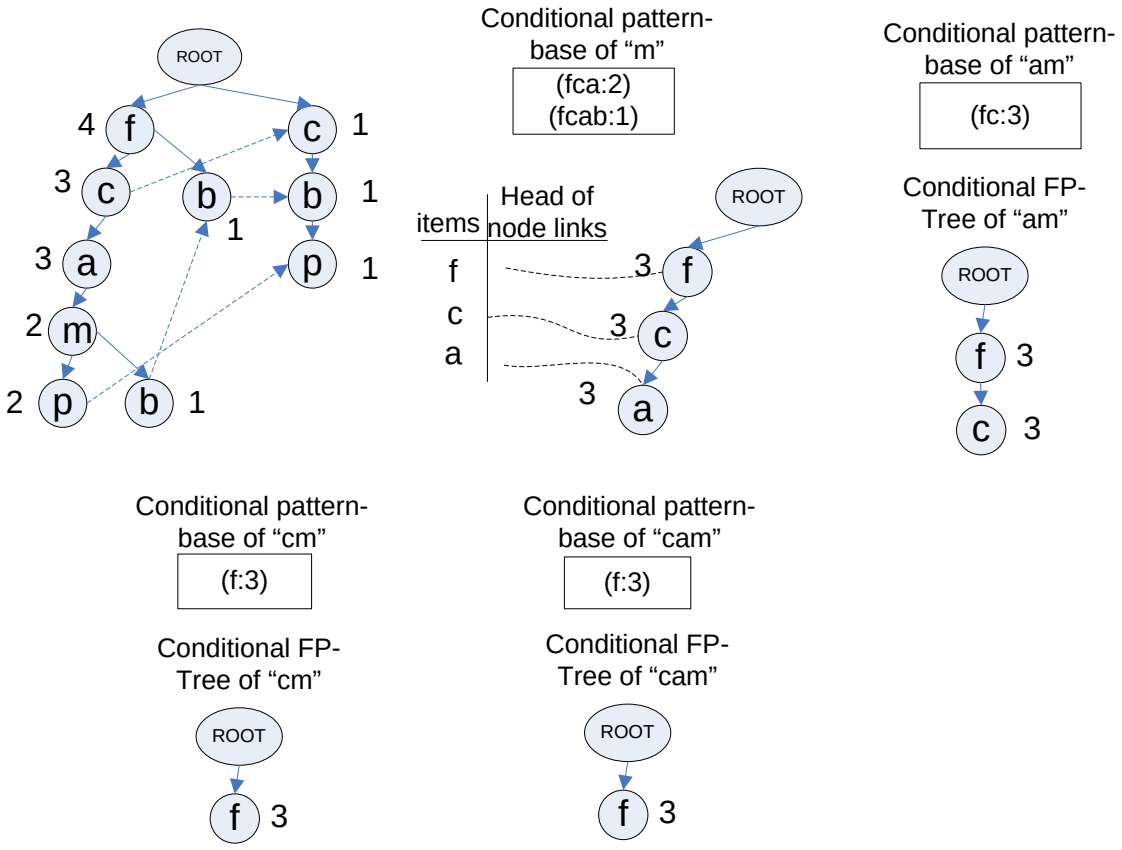

Figure 4: Frequent pattern generation

Since b’s conditional pattern-base (fca:1), (f:1), (c:1) generates no frequent items, we are done with b.
Node a: yields (a:3) and one subpattern base (f c:3). Therefore, a’s set of frequent patterns can be generated by taking their combinations. By concatenating them with (a:3), we get (fa:3),(ca:3),(fca:3).
Node c: c results in (c:4) and one subpattern-base (f:3), therefore c’s frequent pattern is (fc:3).
Node f: yields (f:4) but no conditional pattern-base.

3.2 Scalability Issues

The FP-tree can encourage a divide and conquer approach to data mining. The hope is that the FP-Tree will be a compressed representation of the database. But if the FP-Tree does not fit into main memory it can be broken into subtrees until the tree fits into main memory. FP-Trees however do not do well with short patterns, because they have too many nodes and edges. A real world example of a database with long patterns is a database that tracks Financial Information (i.e. stock prices) over a long period of time.

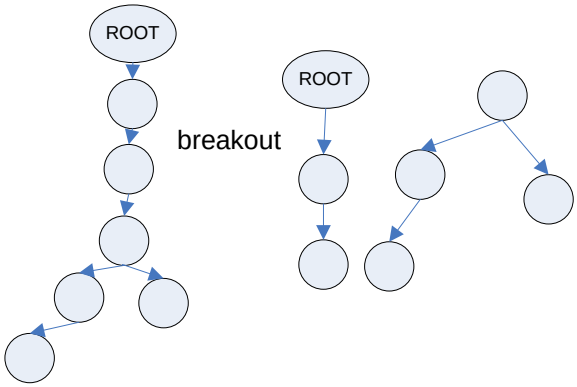

Figure 5: Divide and conquer approach

3.3 Divide and Conquer

As you go through the above algorithm, it should be clear that it is very easy to mine an FP-tree that has a single branch. Just print out all the (proper) subsets of items contained in that FP-tree to get the frequent sets.

This idea can be generalized as follows. If an FP-tree begins with such a lone branch, it can be broken into two FP-trees and the patterns constructed from them separately can be put back together. This allows for a much greater level of parallelism and scalability as mentioned above. (See figure 5)

References

[1] Jiawei Han, Jian Pei, Yiwen Yin, and Runying Mao, Mining Frequent Patterns without Candidate Generation: A Frequent-Pattern Tree Approach, *Data Mining and Knowledge Discovery*, Volume 8, Issue 1, pages 53-87, January 2004