FP-growth distilled

Arnau Abella
Universitat Politècnica de Catalunya

April 28, 2020

1 Introduction

Apriori-like candidate set generation-and-test approach is based on the *anti-monotone Apriori heuristic: if any length k patterns is not frequent in the database, its length* $k+1$ *super-pattern can never be frequent.* The essential idea is to iteratively generate the set of candidates patterns of length $k+1$ from the set of frequent-patterns of length k (for $k \geq 1$), and check their corresponding ocurrence frequencies in the database.

The *Apriori* heuristic achieves good performance gained by (possibly significantly) reducing the size of candidate sets. However, in situations with a large number of frequent patterns, long patterns, or quite low minimum support thresholds, an *Apriori*-like algorithm may suffer from the following: it is costly to handle 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 length-2 candidates and accumulate and test their occurrence frequences. Moreover, to discover a frequent pattern of size 100, such as $\{a_1, \ldots, a_{100}\}$, it must generate $2^{100-2} \approx 10^{30}$ candidates in total.

Mining Frequent Patterns without Candidate Generation: A Frequent-Pattern Tree Approach [HPYM00] proposes a novel frequent-pattern tree (FP-tree) structure, which is an extended prefix-tree structure for storing compressed, crucial information about frequent patterns, and develops an efficient FP-tree-based mining method, *FP-growth*, for mining *the complet set of frequent-patterns* by pattern fragment growth.

Efficiency of mining is achieved with three techniques: (1) a large database is compressed into a condensed, smaller data structure, *FP-tree* which avoids costly, repeated database scans, (2) the FP-tree-based mining adopts a pattern-

fragment growth method to avoid the costly generation of a large number of candidate sets, and (3) a partitioning-based, divide-and-conquer method is used to decompose the mining task into a set of smaller tasks formining confined patterns in conditional databases, which dramatically reduces the search space.

2 Frequent-pattern tree

Definition (FP-tree). A *frequent-pattern tree* is a tree structure defined below.

1. It consist of one root labeled as *"null"*, a set of item-prefix subtrees as the children of the root, and a *frequent-item-header table*.
2. Each node in the item-prefix subtree consists of three fields: *item-name*, *count*, and *node-link*, where *item-name* registers which item this node represents, *count* registers the number of transactions represented by the portion of the path reaching this node, and *node-link* links to the next node in the FP-tree carrying the same item-name, or `null` if there is none.
3. Each entry in the *frequent-item-header table* consist of two fields, (1) *item-name* and (2) *head of node-link* (a pointer pointing to the first node in the FP-tree carrying the *item-name*).

Analysis. The FP-tree construction takes exactly two scans of the transaction database: The first scan collets the set of frequent items, and the second scan constructs the FP-tree. The cost of inserting a transaction T into the FP-tree is $\mathcal{O}(freq(T))$, where $freq(T)$ is the set of frequent items in T.

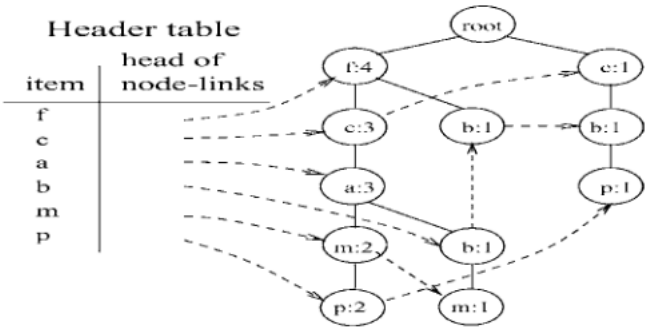

Figure 1: *FP-tree.*

The following *lemmas* are necessary to proof that an *FP-tree* is a compact and efficient data structure and encodes the same information as the original database i.e. it is sufficient scanning the *FP-tree* to find the complete set of frequent itemsets of the database DB.

Lemma 2.1. *Given a transaction database DB and a support threshold ξ, the complete set of frequent item projections of transactions in the database can be derived from DB's FP-tree.*

Lemma 2.2. *Given a transaction database DB and a support threshold ξ. Without considering the (null) root, the size of an FP-tree is bounded by $\sum_{T \in DB} |freq(T)|$, and the height of the tree is bounded by $\max_{T \in DB}\{|freq(T)|\}$, where $freq(T)$ is the frequent item projection of the transaction T.*

3 Mining frequent patterns using FP-tree

This section describes how to explore the comptact information stored in the *FP-tree*, develops the principles of frequent-pattern growth, and proposes a frequent-pattern growth algorithm, *FP-growth*, for mining the complete set of frequent patterns using the *FP-tree*.

Corollary 3.0.1. *(Pattern growth). Let α be a frequent itemset in DB, B be α's conditional pattern-base, and β be an itemset in B. Then $\alpha \cup \beta$ is frequent in DB if and only if β is frequent in B.*

Collorary 3.0.1 is derived from an extensive chain of properties, proofs and examples that are present in the original paper which I recommend reading.

Collorary 3.0.1 states that mining can be performed by first identifying the set of frequent 1-itemsets in DB, and then for each frequent 1-itemset, constructing its conditional pattern-bases, and mining its set of frequent 1-itemsets in the conditional pattern-base, and so on. This indicates that the process of mining frequent patterns can be viewed as first mining frequent 1-itemset and then progressively growing each itemset by mining its conditional pattern-base, which can in turn be done similarly. By doing so, a frequent k-itemset mining problem is successfully transformed into a sequence of k frequent 1-itemset mining problems via a set of conditional pattern-bases. Since mining is done by *pattern growth*, there is no need to generate any candidate sets in

the entire mining process.

Algorithm 1: FP-growth: Mining frequent patterns with *FP-tree* by pattern fragment growth.

Input : A database DB, represented by FP-tree and a minimum support threshold ξ

Output: The complete set of frequent patterns.

```
1  Procedure FP-growth(Tree, alpha)
2  {
3  if Tree contains a single prefix path then
4      let P be the single prefix-path part of Tree
5      let Q be the multipath part with the top branching node replaced by
         a null root
6      foreach combination(denoted as β) of the nodes in the path P do
7          generate pattern β ∪ α with support = minimum support of
             nodes in β
8      let freq_pattern_set(P) be the set of patterns so generated
9  else
10     let Q be Tree
11 foreach item a_i in Q do
12     generate pattern β = a_i ∪ α with support = a_i.support
13     construct β's conditional pattern-base and then β's conditional
         FP-tree Tree_β
14     if Tree = ∅ then
15         call FP-growth(Tree_β, β)
16     let freq_pattern_set(Q) be the set of patterns so generated
17 return freq_pattern_set(P) ∪ freq_pattern_set(Q) ∪ (freq_pattern_set(P)
     × freq_pattern_set(Q))
18 }
```

This algorithm is proven to be enough to find the complete set of frequent itemsets in transaction database DB [HPYM00]. When the FP-tree contains a single prefix-path, the generation of the complete set of frequent patterns can be partitioned into three portions: the single prefix-path portion P, the multipath portion Q, and their combination, line (17). This is extensively explained in the original paper and it is an essencial optimization of the *FP-growth* algorithm.

4 Experimental evaluation and performance study

FP-growth is compared with the classical frequent pattern mining algorithm *Apriori*, and an alternative database projection-based algorithm, *TreeProjection*. The experiments are pursued on both synthetic and real data sets.

The first experiment (figure 2) is to test the compactness of *FP-trees*. They compared the sizes of the following structures.

- `Alphabetical FP-tree`. It includes the space of all the links. However, in such an FP-tree,the alphabetical order of items are used instead of frequency descending order.
- `Ordered FP-tree`. Again, the size covers that of all links. In such an FP-tree, the items aresorted according to frequency descending order.
- `Transaction database`. Each item in a transaction is stored as an integer. It is simply thesum of occurrences of items in transactions.
- `Frequent transaction database`. That is the sub-database extracted from the original oneby removing all infrequent items.

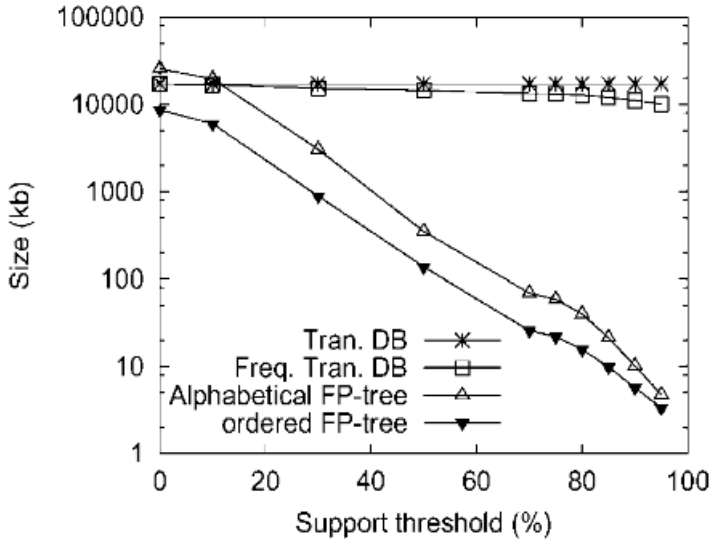

Figure 2: Compactness of *FP-tree* over data set *Connect-4*.

The runtime of *Apriori*, *TreeProjection*, and *FP-growth* on synthetic data set `T10.I4.D100K` as the support threshold decreases from 0.15% to 0.01% is shown in figure 3. *FP-growth* is faster than both *Apriori* and *TreProjection*.

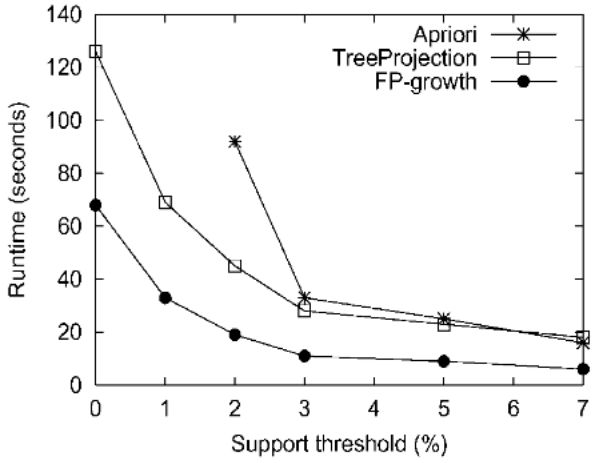

Figure 3: Scalability with threshold over sparse data set.

The advantages of *FP-growth* over *Apriori* becomes obvious when the dataset contains an abundant number of mixtures of short and long frequent patterns. The authors of the paper conducted several more experiments that can be found in the original paper.

5 Conclusions

The presented approach in the paper based on a comptact representation of the database, the *frequent pattern tree (FP-tree)* and the developed mining algorithm based on a *pattern growth* approach offers several advantages over other approaches: (1) highly comptact FP-tree which saves the costly database scans. (2) It applies a pattern growth method which avoids costly candidate generation and test by successively concatenating frequent 1-itemset found in the FP-trees. (3) It applies partitioning-based divide-and-conquer method which dramatically reduces the size of the subsequent conditional pattern bases and conditional FP-tree.

The results of the tests performed show that *FP-growth* outperforms the current candidate pattern generation-based algorithm, in both short and long patterns databases with a huge number of elements.

Since the publication of *FP-growth* method for mining frequent patterns without candidate generation [HPYM00], there have been many subsequent studies on improvements of performance of frequent patterns based on the pattern-growth philosophy [ZF14], as well as extension of the scope of the method to cover other kinds of pattern minin tasks [LCC+12].

References

[HPYM00] Jia Wei Han, Jian Pei, Yiwen Yin, and Runying Mao. Mining frequent patterns without candidate generation: A frequent-pattern tree approach, 2000.

[LCC+12] X. Li, H. Cao, E. Chen, H. Xiong, and J. Tian. Bp-growth: Searching strategies for efficient behavior pattern mining. In *2012 IEEE 13th International Conference on Mobile Data Management*, pages 238–247, 2012.

[ZF14] Zhi Zhang and Qi Fu. Data mining algorithm of frequent probability item based on sliding window. In *Advanced Manufacturing and Information Engineering, Intelligent Instrumentation and Industry Development*, volume 602 of *Applied Mechanics and Materials*, pages 3268–3271. Trans Tech Publications Ltd, 10 2014.