Explore Frequent Itemset Generation methods for Association Rule Mining

Wei Li
School of Electrical Engineering and Computer Science
University of Ottawa
75 Laurier Ave. E, Ottawa, ON K1N 6N5
wli206@uottawa.ca

Abstract

Association Rule Mining is a technique to extract interesting patterns in a large database. A typical application of association rule mining is market basket analysis where relationships between commodities that tend to be bought together can be found in a large transaction database. Association Rule Mining requires generating all frequent itemset from the database. However, the task is intrinsic hard because of both the exponential number of subsets to be considered and the I/O cost of scanning the database. In the project, we study and implement three effective solutions to the frequent itemset generation problem, Apriori[1], FP-Growth[5], and ECLAT[14]. We apply these algorithms to two different data set and compare their efficiency. The result demonstrates that FP-Growth and ECLAT are comparably efficient while Apriori is much slower.

1 Introduction

Association Rule Mining is a technique to identify interesting relations(or patterns) between variables in a large database. Typical usage of association rule mining is market basket analysis. Each day commercial corporations accumulate a large amount of transaction data. A transaction data instance can be represented as

$$tid : \{item_1, item_2, \ldots, item_n\}$$

where tid is a unique id associate with the transaction, and the row stores commodities that were sold in the transaction. Retailers are interested to discover patterns within the database, one of the patterns is association rules represented in forms such as:

$$X : \{item_{p_1}, item_{p_2}, \ldots, item_{p_n}\} \rightarrow Y : \{item_{r_1}, item_{r_2}, \ldots, item_{r_m}\}$$

where X and Y are disjoint subset of the set of all commodities. The rule, mined from the transaction database, represents a strong relationship of subsets of the retailers' inventory such that customers who buy goods in X frequently also tend to buy goods in Y. By using the association rule, retailers can modify their marketing strategies like pricing, discount or inventory management to advance their selling. The application of association rule mining in commercial is called **Market Basket Analysis**.

From Market Basket Analysis, Association Rule Mining has been generalized to a wide range of applications including Bioinformatics, Medical Sciences, Data Science, Natural Language Process. We limit the discussion of the project to Market Basket Analysis.

One problem is that the association pattern discovered from a large database may not be reliable. For one thing, some relationships discovered from the database may be too low-frequent that they should be reduced to accidental events. Even though the relationship is strong, retailers are not interested in promoting low-frequent items. For another thing, a strong relationship between two subsets of items may due to the fact that the second set is generally in high demand. As a result, two frequency-based

metrics, $Support$ and $Confidence$, are used in generating association rules. To compute the two metrics, we need to count the frequency of subsets of items occurring in the transaction database. We will concretely discuss the problem and terms in Section 3.1.

Relying on the two metrics to generate association rules, the problem of Association Rules Mining is often separated into two sub-tasks: 1) generate all subset of items whose frequency exceeds $minsup$, which is a frequency threshold. 2) using frequent itemsets discovered from the last step to generate all association rules that satisfy Support and Confidence constraints.

Solving the first task can be very computationally expensive. Given a set of items with a cardinality of k, there can be $2^k - 1$ subsets to be considered excluding the empty set, letting alone the cost to scan the database for each subset to find its frequency. Numerous Frequent Itemset Generation algorithms have been proposed since the first approach Apriori [1] published in 1993. Exhaustive Methods proposed from the 2000s have been so efficient that few faster algorithms have been found and many researchers have considered the problem as solved (in sequential view). Recent works in the field focus on solving the problem in different parallel architectures, or from heuristics points of view.

Given the frequent itemset generated from the first task, the second task is comparatively easy. Any frequent itemset X and its subsets must satisfy the minimal support $minsup$. Association rules can be generated by separate X into two non-empty subsets X' and $X - X'$ such that they satisfy the minimal confidence $minconf$. If the frequency of all frequent itemsets has been stored, the confidence of each itemset can be directly computed. As we only need to consider the set of all frequent itemset without the need to scan the database, the amount of computation is much smaller.

In the project, we implement and compare three frequent itemset generation algorithms Apriori[1], FP-Growth[5] and ECLAT[14]. We use a realistic Groceries dataset and a synthetic, more challenging data set in the experiment. The rest of the report is organized as 1) related works will be discussed in Section 2. 2) The problem and methods in the project will be discussed in Section 3. 3) Experiments setup and results will be discussed in Section 4.

2 Related Work

The Association rule mining has been solved in two perspectives: sequential algorithms or parallel algorithms; exhaustive algorithms or heuristics-based algorithms. The first association rule mining algorithm Apriori [1] was proposed in 1993 in the context of market basket analysis. Over the years there is a shift of interest from exhaustive to heuristics-based algorithms, and from sequential algorithms to parallel algorithms.

In association rule mining, exhaustive search methods can extract any existing pattern in data. On the other hand, heuristics-based algorithms may potentially miss some patterns since they do not guarantee that important solutions will not be ignored. The motivation to explore with heuristics-based solutions is that heuristic search widely reduces the runtime.

There also exist many works that apply association rule mining algorithms to other fields of studies, which also includes many real-life applications. Also, many variations of the Association Rule Mining problem have also been proposed and studied. For example, the idea of mining frequent itemset can be extended to mining itemsets that are infrequent, or absent [12]. Another problem is High Utility Itemset Mining [4] , where items are not considered equally important, some items are of more utility to the party of interest.

2.1 Sequential Exhaustic Algorithms

Apriori ([1]) was the first proposed solution to association rules mining in 1993, which uses level-wised breadth-first search and pruning in subsets. The main idea behind the algorithm is pruning the search tree based on the apriori property of frequent itemset, which states that any subset of a frequent itemset should also be frequent. After Apriori, many Apriori-based variations had been proposed but did not achieve significant improvement. For example, the author of Apriori proposed AprioriTid and AprioriHybrid [2] which improved the original algorithm. Partition algorithm [11] aims at improving frequent itemsets generation efficiency by reducing the number of disk I/Os during scanning the database. The idea is to split data into chunks and solve each smaller problem, which

later becomes the basis of parallel algorithms. DHP (Direct Hashing and Pruning) algorithm [8] improve Apriori by introducing a hashing data structure.

A key algorithm that differs from Apriori and achieved a significant performance breakthrough is FP-Growth ([5]). The method uses a novel frequent pattern tree (FP-Tree) structure to store pattern information. Unlike the generation-and-test paradigm of Apriori, FP-growth extracts frequent itemsets directly from FP-trees. FP-Growth achieved an order of magnitude faster than Apriori. At the same time ECLAT (Equivalent CLAss Transformation) algorithm [14] was proposed in which vertical database and Depth First Search was used. In a vertical database, each item or itemset is associate with a list of transactions in which it appears. The algorithm generates frequent itemsets by intersecting transaction lists and reduces the memory overhead by separating a problem through equivalent classes. dECLAT [15] achieved several orders of magnitude efficiency with respect to ECLAT from the same author. LCM algorithm [13] was proposed at about the same time. The two algorithms are very efficient such that few latter exhaustive algorithms outperform the two algorithms.

2.2 Sequential Heuristics Algorithms

The main advantage of heuristics methodology in association rules mining is that the runtime is almost constant for whatever data. However, heuristics-based algorithms lack the guarantee that the whole search space is analyzed, which is potentially a requirement in many fields.

([7]) firstly used a genetic algorithm to solve the problem. Some representative algorithms along the timeline are Alatas [3], QuantMiner [10], G3PARM [9], and G3P-Quantitative [6]. Heuristics methods used in Association Rules Mining are mostly genetic algorithms.

3 Methodology

We formalize the frequent set generation problem and association rule generation problem in Section 3.1 and Section 3.2. Three frequent itemset generation methods that are used in the project are Apriori [1], FP-Growth [5] and ECLAT[14], and are discussed respectively in Section 3.3, Section 3.4 and Section 3.5.

3.1 Frequent Itemset Generation Problem

In this section we define terminologies in the frequent itemset generation problem. Let $I = \{i_1, i_2, \ldots, i_n\}$ be the set of all unique items in a market basket database. An **itemset** X is a subset of I, in other words, $X \in I^2$. **k-itemset** is defined as set of itemsets with cardinality k, i.e. $\{X | X \in I^2 \text{ and } |X| = k\}$. A **transaction database** Ω contains sets of itemset, denoted as R, with a unique identifier tid

$$\Omega = \{R_1, R_2, \ldots, R_{tid}, \ldots\}$$

We define a function **support count** δ that takes an itemset X and a transaction database Ω, and return the number of times (frequency) X occurs as a subset in Ω

$$\delta(X, \Omega) = |\{R_i | X \subseteq R_i \text{ and } R_i \subseteq \Omega\}|$$

An itemset X is $frequent$ if its support count $\delta(X, \Omega)$ is greater than a user-defined minimum support ($minsup$).

Frequent itemsets must satisfy the **Apriori Property**, that any subset of a frequent itemset must be frequent.

Given a set of n items $|I| = n$, there can be potentially 2^n itemsets. The problem of Frequent Itemset Generation is to generate all frequent itemsets, i.e. itemsets whose support count $\delta(X, \Omega) > minsup$.

3.2 Association Rules Mining Problem

Given two mutual exclusive itemsets X and Y, a confidence function φ is defined as

$$\varphi(X, Y, \Omega) = \frac{\delta(X \cup Y, \Omega)}{\delta(X, \Omega)}$$

Given a user-defined minimum support $minsup$ and minimum confidence $minconf$, X and Y form an association rule $X \rightarrow Y$ if

$$\delta(X \cup Y, \Omega) \geq minsup \text{ and } \varphi(X, Y, \Omega) \geq minconf$$

The problem of association rules mining is to generate all possible association rules from Ω.

Let $X' = (X, \delta(X, \Omega))$ be a tuple that stores frequent itemset X_i and its support count. Assume all frequent itemsets have been generated and stored in set $F = \{X_i' | X_i' \text{ is a frequent itemset tuple}\}$, then the Association Rules Mining problem is straightforward: given a frequent itemset X, we find two non-empty, mutual exclusive subset of X as Y and $X - Y$ such that $\varphi(Y, X - Y, \Omega) \geq minconf$, and we generate association rule $Y \rightarrow X - Y$ accordingly. The rule must automatically satisfy the minimum support $minsup$ because of the Apriori property. Also, the confidence function $\varphi(\ldots)$ can be conveniently computed as all the support counts of frequent itemsets are stored in F.

3.3 Apriori

In Section 3.3, Section 3.4 and Section 3.5 we discuss Apriori, FP-Growth and ECLAT, three algorithms that solve Frequent Itemset Generation Problem.

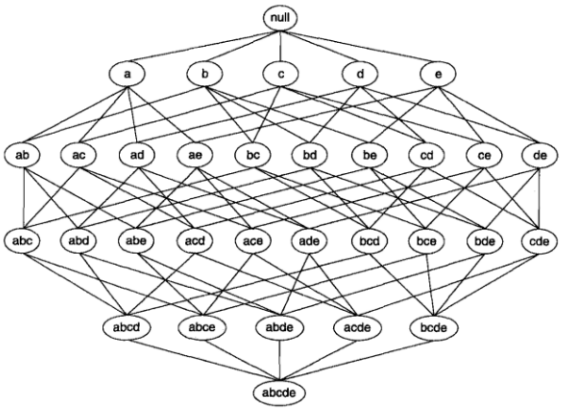


Figure 1: Compare EfficientNet with other image classifier

The power set of a set can be represented as a lattice (Figure 1). Apriori algorithm relies on the Apriori Property of frequent itemset, and conducts a level-wised breadth-first search in the power set lattice. The Apriori property can also be stated as the anti-monotonic property of support count, that the support count of any superset of a set must not exceed the support count of the set: for all itemsetX, Y such that $X \subseteq Y$, $\delta(Y, \Omega) \leq \delta(X, \Omega)$. With the property, Apriori prune early in

small itemsets and only branch from frequent k-itemets to (k+1)-itemsets. We use F_k to denote all frequent k-itemsets. Algorithm 1 describes Apriori.

Algorithm 1: Apriori(Ω, $minsup$)

Result: F

```
1  GLOBAL VAR, estimate;
2  k ← 1;
3  F_1 ← all frequent 1-itemsets;
4  F ← {(f, δ(f)) | f ∈ F_1};
5  while F_k ≠ φ do
6  |  k ← k + 1;
7  |  C_k ← candidate(F_{k-1}, ...);
8  |  for R_i ∈ Ω do
9  |  |  D_i ← {c | c ∈ C_k and c ⊆ R_i};
10 |  |  for d ∈ D_i do
11 |  |  |  δ(d) ← δ(d) + 1;
12 |  |  end
13 |  end
14 |  F_k = {c | c ∈ C_k and δ(c) ≥ minsup};
15 |  F ← F ∪ {(f, δ(f)) | f ∈ F_k};
16 end
17 return F;
```

Apriori generates F which stores all frequent itemsets and their support count. a, b, c, d are used to denote item or itemsets in Algorithm 1.

- In [2-4] the algorithm generates all frequent 1-itemsets and initializes F with F_1.
- Then the algorithm iteratively uses frequent (k-1)-itemset to generate k-itemset candidates C_k [6-7]. Two variation of the $candidate(\ldots)$ function are considered. The first one is union a frequent (k-1)-itemset with a frequent 1-itemset such that the item in F_1 does not appear in the itemset in F_{k-1}. The second one is union two ordered frequent (k-1)-itemsets F_{k-1} such that the first $(k-2)$ item are the same (i.e. only differ from the last item).
  - $candidate1(F_{k-1}, F_1) = \{c = a \cup \{b\} |\ a \in F_{k-1}$ and $(\{b\} \in F_1 \wedge b \notin a)\}$
  - $candidate1(F_{k-1}) = \{c = a \cup b |\ a, b \in F_{k-1}$ and $ordered(a[0 : k-2]) = ordered(b[0 : k-2])\}$

  The two implementations are both complete but they may differ in efficiency.
- To count the support of itemsets in the candidate set, the algorithm scans Ω once. For every record R_i in Ω, the algorithm determines itemsets in C_k that appear also in R_i, and add their support count by 1. [8-13].
- After a complete scan, the algorithm generates frequent k-itemset F_k and adds them to F [14, 15]. When frequent k-itemsets are selected as candidates for the next iteration, the search tree is also pruned. When no k-itemset is generated the algorithm terminates.

3.4 FP-Growth

In Apriori, the algorithm needs to scan the whole transaction database k times. In the worst case, n equals the number of unique items in the transaction database. Although n scans are not likely, as the average length of transactions increases, the I/O cost of database scanning goes up significantly. Also, in practice, a transaction database is large and cannot be load into a single memory.

The idea of FP-Growth [5] is to use a data structure named FP-tree to compress and store information in the transaction database, and generate frequent itemsets directly from FP-tree. To construct the FP-tree, the algorithm only needs to scan the transaction database few times and map each transaction into a path; Each node in FP-tree is an item that may belong to several transactions or several paths, so there is a frequency count that corresponds to each node; In FP-Growth all itemsets are ordered in alphabetical or integer order. Since there are many transactions that share overlapping prefixes, the paths will also overlap. The more overlapping occurs in the database, the more efficient FP-tree is to

compress the information. In the implementation, the FP-tree is generated according to support count rank order so frequent items are closer to the root. Since an item's nodes may occur several times as root to different subtrees or as different leaves, there is a list of pointers that connects nodes that correspond to the same item.

To generate frequent itemsets from FP-tree, FP-growth use a frequency-rank-based bottom-up search the tree, and it also uses a divide-and-conquer strategy. For example, it considers frequent itemsets end with the least ranked item at first, then increases the suffix length to reduce the size of subtrees.

Assume there are n frequent items in the whole problem ordered in decreasing order of items' frequency.

$$I = \{p_1, p_2, \ldots, p_n\}$$

and all transaction are also ordered in decreasing order of items' frequency. p_n is the least frequent frequent-item. The algorithm firstly generates all frequent itemsets end with p_n, then frequent itemsets end with p_{n-1}... and finally the 1-itemset $\{p_0\}$. To generate all frequent itemsets end with p_n, the FP-Growth algorithm reduce the FP-tree:

1. From the initial FP-tree, generate a subtree ST_{p_n} that contains p_n in leaves, count the total support count of p_n nodes to determine if it is frequent

2. Convert the subtree into a conditional FP-tree CFT_{p_n} condition on p_n by

   - a) decrease support count of nodes in paths that do not include p_n, such that the support count of leaves are consistent to nodes in paths
   - b) delete p_n from leaves
   - c) delete non-frequent items from the subtree in b)

3. Then the algorithm considers all 2-suffix itemsets that end with p_n

$$\{(p_{n-1}, p_n), (p_{n-2}, p_n), \ldots, (p_1, p_n)\}$$

by recursively call the process over CFT_{p_n}.

3.5 ECLAT

ECLAT(Equivalent CLAss Transformation) [14] differ from Apriori in two aspects: 1) it uses the vertical database to store transaction data. 2) it conducts a depth-first search instead of a level wised breadth-first search in the itemset lattice. The aim of ECLAT is, like FP-growth, to reduce the I/O cost of scanning the database and solve the problem more efficiently by divide-and-conquer.

In vertical database, each line of data store set of transaction id (tid) that an item appears in. One line of data can be represented as:

$$TID(item_i) = \{tid^1_{item_i}, tid^2_{item_i}, \ldots, tid^m_{item_i}\}, |TID(item_i)| = m$$

TID function can also be generalized to itemset. The function takes an itemset X and returns a set of tid in which X appears as a subset in the corresponding transaction.

If an itemset X can be divided into disjoint subsets $X = \bigcup_{i=1}^{k} X_k$, then $TID(X)$ can be calculated by intersecting TID sets of all subsets.

$$TID(X) = \bigcap_{i=1}^{k} TID(X_k)$$

and support count of X can be easily calculated as the cardinality of $TID(X)$. The vertical database is implemented as one-hot vector for quicker intersection.

ECLAT is based on the idea that finding small frequent itemsets and merge them into larger itemsets, and keeps all intermediate TID sets for reference. As the itemset grows, the length of TID sets shrinks and eventually leads to a non-frequent itemset. One problem is that the initial and intermediate TID sets may be too big to fit in memory space. So ECLAT uses equivalent class transformations to partition the large initial problem into independent smaller problems and solve them separately. We

use the dictionary order of commodities' names to index items. We use prefix-based partition during the recursive search.

Given equivalent partitions, ECLAT searches in each partition in itemset lattice with bottom-up depth-first order. Like Apriori, ECLAT also use the Apriori property to search and backtrack.

In the algorithm, we use F_k^X to refer frequent k-itemset with prefix X. TID_k is a set of all one-hot $TID(X)$ of frequent k-itemsets.

$$TID_k = \{TID(X) | \ X \in F_k\}$$

and there is a mapping from $X \in F_k$ to $tid_X \in TID_k$.

$$tid_X = TID(X) \ , X \in F_k \text{ and } tid_X \in TID_k$$

The recursive call in ECLAT is given in Algorithm 2: ECLAT. The global variable **RESULT*** contains frequent itemsets and their support count.

Algorithm 2: ECLAT(F_k, $minsup$)

Result: F

```
1  GLOBAL VAR, RESULT*;
2  if F_k is None then
3  |   F_k ← F_1;
4  |   TID_k ← TID_1;
5  end
6  for X ∈ F_k do
7  |   sort X in place ;      // X ∈ F_k are ordered set, F_k is also ordered according to
   |    prefix
8  end
9  while F_k is not empty do
10 |   X ← F_k.pop();
11 |   tid_X ← TID(X);
12 |   save X and sum(tid_X) to RESULT*;
13 |   F_{k+1}^X ← φ;
14 |   for Y in F_k do
15 |   |   Z ← X ∪ Y;
16 |   |   tid_Y ← TID(Y);
17 |   |   tid_Z ← tid_X × tid_Y;
18 |   |   if sum(tid_Z) ≥ minsup then
19 |   |   |   Z ← sort(Z);
20 |   |   |   F_{k+1}^X ← F_{k+1}^X ∪ Z;
21 |   |   end
22 |   end
23 |   ECLAT(F_{k+1}^X, minsup);
24 end
25 return RESULT*;
```

If it is the first call, ECLAT get all frquent 1-itemsets and their TID set, and store the result to F_k and TID_k[2-5].

Then the algorithm sort all itemsets in F_k in increasing order (all itemsets and F_k are ordered)[6-8].

Then the algorithm sequentially pop itemsets X and tid_X in F_k. It firstly save the frequent set to result, then union the itemset with other frequent itemsets and evaluate the resulting itemset Z and tid_Z. When tid_Z satisfies $minsup$, i.e. Z is frequent, Z and tid_Z are added to F_{k+1}^X for the resursive call ECLAT(F_{k+1}^X, $minsup$). [9-24]

4 Experiment

4.1 Environment

System: Windows 10
Hardware: Intel core I7 9700; 8+8 GB speed 2667;
Language: Python 3.8.3

4.2 Dataset

In the experiment we consider two data set.

DatasetA: DatasetA is a small, realistic groceries data set. The raw data set consists of 38765 line transactions of one commodity. After grouping transaction date and member, the data set becomes 14963 transactions. There are 167 different commodities in the data set. We map commodities name (English string) to positive integer in alphabetical order. Some transactions involve repetitive item, in which case we discard them to make each item only occurs once in a transaction. The average length of transactions is 2.54.

DatasetB DatasetB is a more challenging synthetic databases with 540455 transactions. The data set is made of positive integers so there is no need to do pre-processing. There are 2603 unique integers in the data set and the average transaction length is 4.37.

4.3 Minimum support and Complexity

In the experiment, we compare the three algorithms' efficiency of solving frequent itemset generation problem. We measure the efficiency in running time(s).

Since different data set have different transaction length, we use proportional support function. The support count of itemset X is

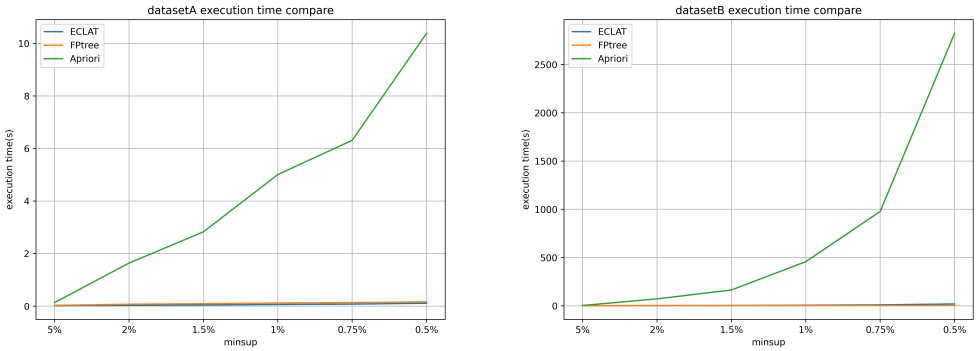
$$support = \frac{\delta(X)}{|\Omega|}$$

We experiment six different $minsup$ values.

$$minsup \in \{5\%, 2\%, 1.5\%, 1\%, 0.75\%, 0.5\%\}$$

A natural relationship between running time and $minsup$ is that: as the $minsup$ decrease, more itemsets in the itemset lattices becomes frequent (being activated). As a result, more routes or branches need to be considered during the generation, which may result in a longer search time.

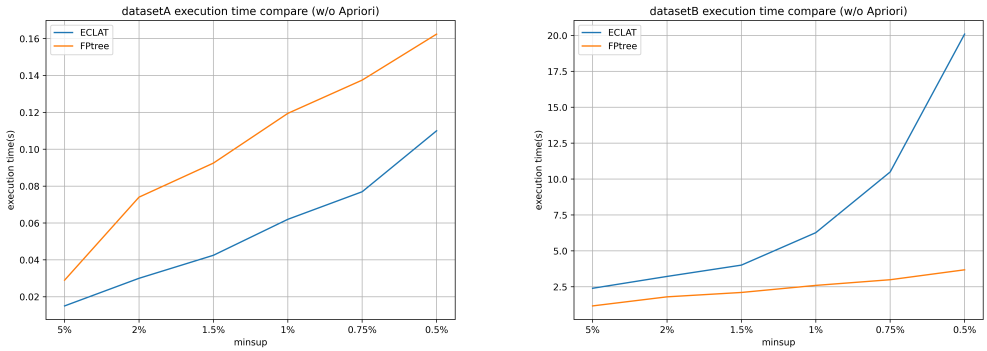
4.4 Experiment Result



(a) Comparing three algorithms on DatasetA

(b) Comparing three algorithms on DatasetB

Figure 2: Result with Apriori



(a) Comparing ECLAT and FP-tree on DatasetA (b) Comparing ECLAT and FP-tree on DatasetB

Figure 3: Result without Apriori

Experimental results are given in Figure 1 and Figure 2. In Figure 1 we compare the execution time of the three algorithms together. The result shows that ECLAT and FP-Growth were both an order of magnitude faster than the Apriori algorithm. In Figure 2 we exclude Apriori's results to compare ECLAT and FP-Growth. It is observed that ECLAT was faster in the smaller datasetA, but slower than FP-Growth in the more challenging datasetB. In datasetB, when we reduced the $minsup$, FP-Growth's running time grew almost linearly, but ECLAT grew much quicker.

5 conclusion

In the project, we explored the association rule mining problem and the frequent set generation problem. We learned three algorithms, Apriori, FP-Growth and ECLAT to generate all frequent itemsets within a transaction database, and implemented them to solve two frequent itemset generation data sets. The result demonstrates that ECLAT and FP-Growth are comparably effective, and an order of magnitude faster than Apriori.

References

[1] R. Agrawal, T. Imielinski, and Arun N. Swami. "Mining association rules between sets of items in large databases". In: *SIGMOD '93*. 1993.

[2] R. Agrawal and R. Srikant. "Fast Algorithms for Mining Association Rules in Large Databases". In: *VLDB*. 1994.

[3] Bilal Alatas and E. Akin. "An efficient genetic algorithm for automated mining of both positive and negative quantitative association rules". In: *Soft Computing* 10 (2006), pp. 230–237.

[4] Wensheng Gan et al. "A Survey of Utility-Oriented Pattern Mining". In: *IEEE Transactions on Knowledge and Data Engineering* 33 (2021), pp. 1306–1327.

[5] Jiawei Han, J. Pei, and Y. Yin. "Mining frequent patterns without candidate generation". In: *SIGMOD '00*. 2000.

[6] J. M. Luna et al. "Reducing gaps in quantitative association rules: A genetic programming free-parameter algorithm". In: *Integr. Comput. Aided Eng.* 21 (2014), pp. 321–337.

[7] J. Mata, J. L. Alvarez, and J. Riquelme. "Mining Numeric Association Rules with Genetic Algorithms". In: 2001.

[8] J. S. Park, M. Chen, and Philip S. Yu. "An effective hash-based algorithm for mining association rules". In: *SIGMOD '95*. 1995.

[9] C. Romero et al. "Association rule mining using genetic programming to provide feedback to instructors from multiple-choice quiz data". In: *Expert Syst. J. Knowl. Eng.* 30 (2013), pp. 162–172.

[10] Ansaf Salleb-Aouissi, Christel Vrain, and Cyril Nortet. "QuantMiner: A Genetic Algorithm for Mining Quantitative Association Rules". In: *IJCAI*. 2007.

[11] Ashok Savasere, E. Omiecinski, and S. Navathe. “An Efficient Algorithm for Mining Association Rules in Large Databases”. In: *VLDB*. 1995.

[12] Ashok Savasere, E. Omiecinski, and S. Navathe. “Mining for strong negative associations in a large database of customer transactions”. In: *Proceedings 14th International Conference on Data Engineering* (1998), pp. 494–502.

[13] T. Uno, Masashi Kiyomi, and H. Arimura. “LCM ver. 2: Efficient Mining Algorithms for Frequent/Closed/Maximal Itemsets”. In: *FIMI*. 2004.

[14] Mohammed J. Zaki. “Scalable Algorithms for Association Mining”. In: *IEEE Trans. Knowl. Data Eng.* 12 (2000), pp. 372–390.

[15] Mohammed J. Zaki and K. Gouda. “Fast vertical mining using diffsets”. In: *KDD ’03*. 2003.