Rare Pattern Mining from Noisy Data

by

Connor Clark James Hryhoruk

A thesis submitted to the Faculty of Graduate Studies of
The University of Manitoba
in partial fulfillment of the requirements for the degree of

Master of Science

Department of Computer Science
The University of Manitoba
Winnipeg, Manitoba, Canada

August 2023

Thesis advisor
Dr. Carson K. Leung

Author
Connor Clark James Hryhoruk

Rare Pattern Mining from Noisy Data

Abstract

Data mining is the process of discovering previously unknown, yet useful, information from data. Data is often large, requiring automated forms of analysis to extract interesting patterns and relationships. To add to this problem, interesting information may be lost when noise is introduced into data. Existing approaches apply lenience to support to accommodate the impacts of noise on frequent patterns. Whilst past research provides improvements to mining within noisy datasets, limited or no work has considered support lenience in relation to rare pattern mining. Rare patterns, as by their name, do not commonly occur. Discovering rare patterns has a variety of applications in areas such as medicine, market analysis, and outlier detection, where commonly occurring events are often uninteresting as they are already well-known and thus do not provide novel insight. I introduce the mining of approximately rare itemsets to discover rare patterns with data in the presence of noise. To mine approximately rare itemsets, an ARI-growth algorithm is proposed. Computational results show ARI-growth is an order of magnitude faster than an Apriori approach. Memory consumption shows similar trends, although, is outperformed by Apriori in one tested case. Approximation with ARI-growth is applicable for rare core pattern recovery.

Table of Contents

Acknowledgements

I am grateful to have the rare opportunity to have Dr. Carson K. Leung as my supervisor and mentor. Dr. Leung is the type of person who will support you when you are lost. To accommodate you when in need. To give great advice. To work 100+ hour weeks, and have resilience to distress. To lead by example, whilst having care for students. I cannot imagine having pursued a graduate-level education without him. Dr. Leung provides the best possible resource for students. I appreciate your mentorship and wish you the best in your future well-being.

Thank you to my M.Sc. thesis defence chair, Dr. Avery Miller. Also, thank you to my M.Sc. thesis examination committee members, Dr. Shaowei Wang (Computer Science) and Dr. Lisa Lix (Community Health Sciences). Both Dr. Wang and Dr. Lix have provided valuable time and effort in order to allow this thesis to take place. I am glad to have your support. May the world treat you kindly.

Connor Clark James Hryhoruk

B.C.Sc.(Hons.), The University of Manitoba, 2021

The University of Manitoba

August 21, 2023

This thesis is dedicated to my family.

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Data Mining

Data mining is a non-trivial extraction of implicit, potentially useful, and previously unknown information from data. Mining was first popularized in 1993 by Agrawal et al. [AIS93], growing in interest through the discovery of association rules. An *association rule* is a relationship formed from logical implications of patterns from data. For example, measuring the relationship between commonly purchased groceries such as 'milk' and 'cereal' may be useful for restructuring a supermarket to optimize customer convenience. An association is represented in the form $A \Rightarrow B$, or in other words, the domain item or set of items (itemset) A implies the existence of itemset B. A common measure for the strength of an association is confidence, defined below:

$$Confidence(A \Rightarrow B) = \frac{Sup(A \cup B)}{Sup(B)} \tag{1.1}$$

where $Sup(X)$ for any itemset X denotes the support of an itemset. Support is a measure of the number of occurrences attributed to the itemset. For instance, if 2 customers purchase milk, then milk is said to have a support of 2 (i.e., $Sup(milk) = 2$).

Discovering association rules is a 2-step process. Step 1 is to discover commonly occurring (i.e., frequent) itemsets (i.e., patterns), and Step 2 is to form association rules for each frequent pattern to discover interesting associations. In this thesis, the focus is in regards to Step 1—the process of extracting frequent patterns because this step is more computationally intensive than Step 2. Extracting patterns is a computationally expensive task as any combination of items yields a pattern.

To reduce unnecessary analysis, two fundamental concepts are defined. Firstly, a minimum support threshold (*minsup*) is used to determine what constitutes a frequent pattern. Under the circumstance that a pattern's support is greater than or equal to the *minsup*, the pattern is classified as frequent. Secondly, when evaluating itemsets, it is not necessarily the case where additional items are of interest. More specifically, if any itemset is not frequent, then any additional item to the itemset will also be infrequent. Ignoring additional items is a result of the anti-monotone property.

The *anti-monotone property* states that all proper subsets of a frequent itemset are frequent, or conversely if an itemset is infrequent, all proper supersets are infrequent. A subset is an itemset that is contained within another. For example, $\{A\}$ is a subset of $\{A, B, C\}$, whilst a proper subset is an unequal subset. For this example, there are 7 proper subsets of $\{A, B, C\}$. These 7 proper subsets, together with itself $\{A, B, C\}$, give 8 subsets of $\{A, B, C\}$. Similarly, a superset is the opposite of a

subset; an itemset that contains another. $\{A, B\}$ is a superset of $\{A\}$ since all items $\{A\}$ are contained within $\{A, B\}$. The genius of the anti-monotone property is the allowance for substantial improvements to pruning itemsets that are guaranteed to be uninteresting. For example, let us consider an itemset $\{A, B\}$ that is infrequent by our defined minsup, it is a certainty that $\{A, B, C\}$ is also infrequent. As a result, proper supersets of all infrequent itemsets can be ignored. Below the monotone property is proved (Lemma 1.1), where subsets of a frequent itemset are frequent. Intuitively, anti-monotone must also then be true as a supersets support must be less than or equal to that of any subset.

Lemma 1.1. *Monotone property: Subsets of a frequent itemset are frequent.*

Proof. Proof by counter example: Given a minimum support threshold s, and an itemset $X = \{A_1, A_2, ..., A_n\}$ with $n \geq 1$ that is frequent. Let us suppose there exists an itemset $Y \subset X$ & $|Y| > 0$ that is infrequent. By definition of frequent, $support(X) \geq s$. By definition of support, in a dataset d, there exists exactly k occurrences of interest that contain $\{A_1, A_2, ..., A_n\}$, where $k \geq s$. It follows, $Y \subset \{A_1, A_2, ..., A_n\}$, so there exists at-least k occurrences of interest in d that contain Y. Since $t \geq s$, Y is frequent. A contradiction is reached, and thus for any $n \geq 1$, the monotone property holds. □

Frequent pattern mining is a useful measure for extracting commonly occurring events from big data however, this measure of discovering patterns relies on *exact* precision. Under a dataset with lost records, frequent patterns may no longer be frequent as a result of decreased support. This calls for *imprecise* solutions that

provide lenience to the number of items required in a itemset that contribute to support.

1.2 Mining in Noisy Data

Datasets commonly contain lost, unknown, corrupted, or miss-translated records. As a result, frequent patterns are at risk of being misclassified. Misclassification of frequent patterns occurs due to exact precision requirements of support in frequent pattern mining. For example, given an item $\{A\}$, if noise causes data that contains $\{A\}$ to be lost, then the support of $\{A\}$ is reduced. The support of $\{A\}$ may then fall below the *minsup* and therefore become infrequent when the itemset would have otherwise been frequent. For future reference, in this thesis, the strict definition of support that does not tolerate information change from traditional mining is referred to as *exact mining*, and information change resulting in lost, unknown, corrupted, or miss-translated records as *noise*.

As expected, as noise increases, more frequent patterns are lost. Loss of frequent patterns further accelerates with an increased support threshold. The anti-monotone property of exact mining leads to the pruning of itemsets that would otherwise be frequent. Let us suppose that from noise item $\{A\}$ is lost from transaction Y. In this case, the support of $\{A\}$ is decreased by 1. A 'domino' effect occurs where all supersets of $\{A\}$ that contain another item within transaction Y will also decrease in support. As such, one or more patterns are affected for every 1 item lost by noise. In addition, if any such itemset becomes infrequent, all supersets are also infrequent. Having a form of accurately recovering frequent patterns in the presence of noise is

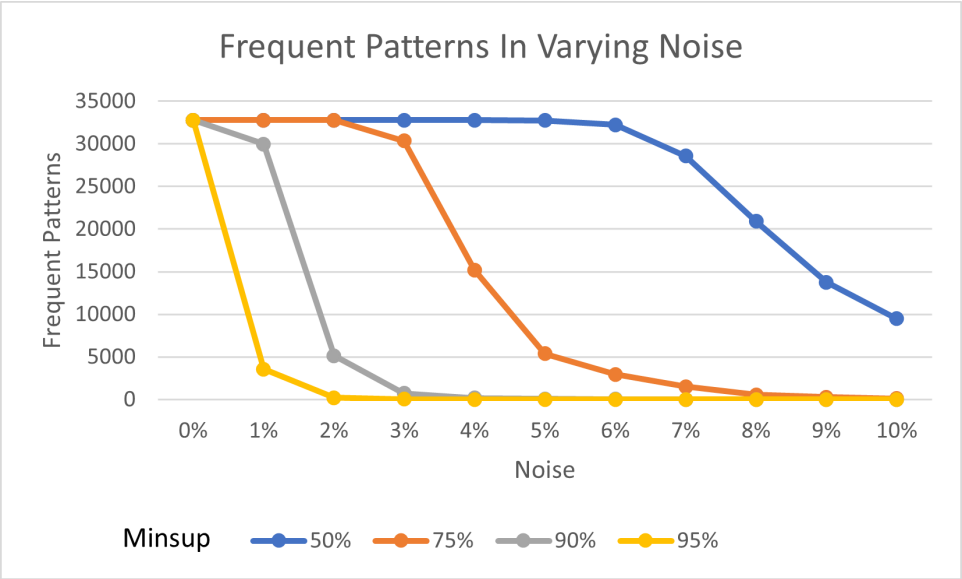

Figure 1.1: Pattern loss as a result of noise. 15×100 all-one matrix.

an important task to ensure meaningful information can be extracted from data.

An application of noise recovery can be found in the healthcare domain. Let us consider international classification of diseases (ICD) codes [WHO] of hospitalizations or visits to a physician in healthcare. ICD codes are a representation of a patient's condition at the date of discharge. ICD codes have been used in many forms of research such as that of association rule mining [BSRD23]. Recent studies that use prediction models from healthcare data do not handle missing values that result in worsened analytical accuracy and capability [NLB+22]. Medical records may contain data entry errors that invalidate a physician visit, or additionally, ICD codes may be miss-classified. With respect to the mining of precise patterns, loss of information results in the reduction of frequency. For instance, a miss-classified ICD code results in the frequency adjustment of two different ICD codes, which, may prevent capturing of an interesting association. As frequency decreases, patterns that are frequent

Table 1.1: Data with missing values.

Transaction ID	Items
0	{A, B}
1	{A, Null}
2	{A, B, C}
3	{B, D, Null}

have the potential to fall below the minimum support threshold, becoming uninteresting or undetected, and additionally, changes to frequency will adjust discriminatory measures of determining interesting associations. Approximation of truly interesting patterns to accommodate for noise may be applied to recover lost information.

One solution to recover itemsets with missing data was proposed in 2001, denoted as mining of fault-tolerant (FT) frequent itemsets [PTH01]. An FT frequent itemset differs from the traditional approach; an itemset no longer requires all items to be present to contribute to frequency. Fault tolerance generalizes the standard *minsup* threshold by including a user-defined error tolerance (δ). Table 1.1 provides an example of a noisy transactional dataset where missing information is labelled as 'N/A'. Let us consider a minimum support threshold of 3, under the circumstance that transaction 1 contained B or transaction 3 contained A; $\{A, B\}$ is a frequent itemset that would not be detected by frequent pattern mining. In the case of FT frequent itemsets, $\{A, B\}$ can be recovered. Estimating the true frequency of a pattern can be referred to as *approximate mining*.

Pattern mining in the presence of noise is computationally expensive. Exact mining utilizes the antimonotone property. The anti-monotone property of exact mining fails when applying lenience to support. As a result, mining is expensive. As data

mining notoriously involves large datasets, improvements to the discovery of patterns without exact matching requirements are an active area of research. In this work, focus is directed towards *rare itemsets*.

In 2019, Borah and Nath [BN19] showed rare pattern mining has increased in research interest with a growing number of algorithms proposed. In practice, in contrast to frequent itemsets, rare itemsets are difficult to evaluate as a result of their low frequency. To elaborate, with traditional frequent pattern mining in order to capture these rare itemsets, the minimum support threshold must be set very low. A low threshold yields a significantly high number of redundant, previously known, and uninteresting patterns; in addition to increased computational constraints. The difficulty of capturing rare itemsets is known as the **rare itemset problem**, motivating algorithmic improvements in rare pattern mining. Under the presence of noise, challenges faced by the rare itemset problem are enhanced further, as itemsets with lower support are highly sensitive to noise from the greater proportion of support loss.

1.2.1 Thesis Statement and Contributions

Here, my **M.Sc. thesis statement** is to mine rare itemsets within an environment that contains noisy data. My **key contributions** include the proposition of discovering rare itemsets in the presence of noise and an improved pattern-growth approach for approximate mining. The following are questions to be explored in this thesis, with respect to discovering patterns in a noisy dataset by applying lenience to support:

1. How can rare itemsets be discovered?

2. Can itemsets that are approximated be discovered efficiently to be applicable in the real world?

3. How does efficiency compare to that of traditional mining techniques?

4. Can approximation be applied to recover lost, yet interesting, rare itemsets?

5. Can the same approximation of itemsets be applied to recover lost, yet interesting, association rules?

1.2.2 Thesis Organization

This thesis is structured as follows: In Chapter 2, a further overview of frequent pattern mining and two popular mining algorithms are provided. Next, rare pattern mining is introduced. Finally, frequent pattern mining in the presence of noise is introduced. In Chapter 3, mining rare itemsets in noise with an applied approximation model and our newly proposed algorithm is introduced. Chapter 4 signifies the evaluation of our proposed technique, where the algorithm is paired against rare pattern mining algorithms in terms of computation, and evaluated in terms of pattern recovery from lost information in the presence of noise. Concluding statements are provided in Chapter 5.

Chapter 2

Background and Literature Review

In this chapter, relevant background material to the M.Sc. thesis is provided. Frequent pattern mining discovers association rules with commonly occurring items in a dataset. In contrast, one may wish to uncover itemsets that do not occur commonly and thus consider rare pattern mining. Lastly, problems with discovering interesting patterns are prominent when a presence of noise exists in a dataset. Patterns generated by both frequent and rare pattern mining techniques may no longer provide truthful or meaningful outcomes. As a result, determining an appropriate technique to discover these relationships with a noisy dataset has had ongoing research. An overview of frequent, rare, and noisy pattern mining is provided.

2.1 Exact Frequent Pattern Mining

Frequent pattern mining uses a transactional database to uncover new knowledge through the extraction of commonly occurring events. A *transactional database*

(TDB) is a list of sets of items. Frequent pattern mining is motivated by market basket analysis [AS+94], where datasets in the retail domain are analyzed to determine associations that can be used to improve sales. For example, through associations, if a strong association between the purchase of milk and cereal is observed, store layout can be adjusted by placing these products not far from one another to maximize customer convenience. The transactional database in this domain includes a list of transactions made by customers, where each transaction contains some set of purchased items. An example TDB is shown in Table 2.1.

Frequent pattern mining is an important area of research for tasks that include but are not limited to; discovering correlations, emerging patterns, artificial intelligence, and associations. Frequent pattern mining is applied in many forms of research such as summarizing large amounts of data in clinical research [YLL+20], counterfeit detection [BBL22], and market analysis to determine customer trends [MEMP21].

Frequent patterns are however expensive to compute for large datasets. In order to accommodate large amounts of data, ongoing algorithmic improvements have been proposed since the 1990s. In this section, two popular algorithms are reviewed: Apriori and FP-growth. For future reference of Chapter 2.1, suppose there exists a transactional database in Table 2.1.

Table 2.1: Transactional database 1.

Transaction ID	Items
0	{A, B, C, E}
1	{A, B, E}
2	{A, B, C, E}
3	{B, C, D}

2.1.1 Apriori Algorithm

Apriori was proposed in 1994 by Rakesh Agrawal [AS+94]. The basis of this algorithm is to create possible combinations of itemsets that could be frequent, and then confirm the results by pruning those that are not frequent. Suppose $minsup = 3$. The algorithm is classified in four stages: initial scan, join step, prune step, and large itemset check.

Stage 1: Initial Scan

First, size-1 itemsets (i.e., itemsets of length [1] 1) are determined through an initial database scan. The process of a database scan is as follows; for each transaction, for each item within the given transaction, increment a numeric value representing the number of times this item has occurred within the database. Once this process is complete, all items with their numeric value (i.e., support) less than our minimum support threshold of 3 are removed. Each itemset that surpasses the minimum support criteria is a size-1 frequent itemset. Results are demonstrated in Table 2.2. After the trimming infrequent itemset $\{D\} : 1$, the resulting itemsets are: $\{A\} : 3$, $\{B\} : 4$, $\{C\} : 3$, and $\{E\} : 3$.

Stage 2: Join Step

Next, using size-1 frequent itemsets, combinations are generated to determine potential frequent itemsets of greater length. Let us denote our current size as k, where initially $k = 1$. Let us denote the set of frequent itemsets of size k as Lk

[1] **Length** is the number of items of an itemset. For an itemset X, this can be denoted as $|X|$. For example, $X = \{A, B, C\}$ is an itemset of length 3; or $|X| = 3$.

(i.e., large-k or frequent-k itemsets), and the set of all potentially frequent itemsets as Ck (i.e., candidate-k itemsets). The set of itemsets C(K+1) can be generated through joins of itemsets Ck. Given two itemsets $\{A\}$ and $\{B\}$, $\{A\} \cup \{B\}$ yields $\{A, B\}$. Joining of itemsets becomes more complex for larger itemsets. For example, merging $\{A, B, D\}$ and $\{C, F, E\}$ can yield a wide variety of size-4 itemsets. While it is possible to generate all possible size-4 itemsets, there is unnecessary computation in generating such candidates due to the anti-monotone property. To resolve this issue, Apriori commonly joins itemsets with k-1 items matching. In this work, a *prefix-join* is used. Prefix-join requires the first k-1 items in a chronologically ordered itemset to be equal. With this optimization, $\{A, B, D\}$ and $\{C, F, E\}$ yields no size-4 itemsets, however, $\{A, B, D\}$ and $\{A, B, E\}$ generates the candidate $\{A, B, D, E\}$. In order for $\{A, B, D, E\}$ to be frequent, all subsets must also be frequent, which implies that a prefix-join must exist in order for the itemset to be frequent, and the itemset is infrequent if there is no applicable prefix-join.

Stage 3: Prune step

The prune step ensures all subsets of a given itemset are frequent. By the anti-monotone property, if this fails, the itemset cannot be frequent. Consider a prefix join between the itemsets $\{B, C\}$ and $\{B, E\}$. While the resulting itemset is $\{B, C, E\}$, it would not be possible for $\{B, C, E\}$ to be frequent if $\{C, E\}$ is not also frequent. In this example, if $\{C, E\}$ is not found, then $\{B, C, E\}$ is discarded.

Stage 4: Large/Frequent step

After the prune step comes the final stage of the algorithm. Given the list of all potentially frequent itemsets, a database scan is performed to count the number of transactions that the itemset takes place within. Any itemset whose count falls below *minsup* is discarded. The algorithm then repeats the join, prune, and large step with a new k value $k = k + 1$ until no frequent itemsets of size k are found.

Example 2.1.1. *Continuing the algorithm through Table 2.1 after the initial database scan with k=1 and minsup* $= 3$*;* $\{A\}$ *and* $\{B\}$ *generates* $\{A, B\}$*.* $\{A\}$ *and* $\{C\}$ *generates* $\{A, C\}$*.* $\{A\}$ *and* $\{E\}$ *generates* $\{A, E\}$*.* $\{B\}$ *and* $\{C\}$ *generates* $\{B, C\}$*.* $\{B\}$ *and* $\{E\}$ *generates* $\{B, E\}$*. Finally,* $\{C\}$ *and* $\{E\}$ *generates* $\{C, E\}$*. A database scan is performed to determine support,* $\{A, C\}$ *and* $\{C, E\}$ *are infrequent. Next,* $\{A, B\}$ *and* $\{A, E\}$ *generates* $\{A, B, E\}$*.* $\{B, C\}$ *and* $\{B, E\}$ *generates* $\{B, C, E\}$*.* $\{B, C, E\}$ *is pruned due to* $\{C, E\}$ *being infrequent. A database scan is performed, with no itemset removed. No size 4 itemsets can be generated with* $\{A, B, E\}$*, and thus the algorithm stops.*

Table 2.2: Size-1 itemsets from database scan.

C1: Initial Scan

Itemset	**Support**
A	3
B	4
C	3
D	1
E	3

L1: Trim Infrequent Items

Itemset	**Support**
A	3
B	4
C	3
E	3

Table 2.3: Remaining itemsets.

C2: Join and Prune Step

Itemset
A, B
A, C
A, E
B, C
B, E
C, E

L2: Frequent C2

Itemset	**Support**
A, B	3
~~A, C~~	~~2~~
A, E	3
B, C	3
B, E	3
~~C, E~~	~~2~~

C3: Join and Prune Step

Itemset
A, B, E
~~B, C, E~~

L3: Frequent C3

Itemset	**Support**
A, B, E	3

Frequent Patterns = $L1 \cup L2 \cup L3$

2.1.2 FP-growth Algorithm

Apriori was a novel algorithm by introducing measures of discovering interesting association rules by returning the set of frequent patterns. Trouble arises when databases are large. Recall, for every set of size-k candidates Apriori generates, a database scan must be performed to determine frequent size-k itemsets. For large datasets, hits database scan hinders performance. In addition, the high number of candidates leads to high memory consumption, and potentially wasted computation as a candidate itemset is not guaranteed to be frequent. As a result, Apriori may not always be a practical solution. Computation time and memory are two constraints demanding improvements in the area of data mining.

To address these constraints, FP-growth was proposed. FP-growth (frequent pattern-growth) is a novel mining technique proposed by Han *et al.,* in 2000 [HPY00]. FP-growth improves upon Apriori by performing two database scans and eliminating candidate generation. FP-growth consists of two stages, setting up a data structure called an FP-tree, and recursively mining the FP-tree. Precise steps in the algorithm are provided below.

Step 1: FP-tree Construction

First, an FP-tree (frequent pattern tree) is constructed. An initial database scan is performed to determine frequent size-1 itemsets. Refer to Section 2.1.1 for details of this initial database scan. Next, frequent size-1 itemsets are ordered by their support in descending order; this ordered set is referred to as a header table. A second database scan is then performed in the following manner; for each transaction, insert

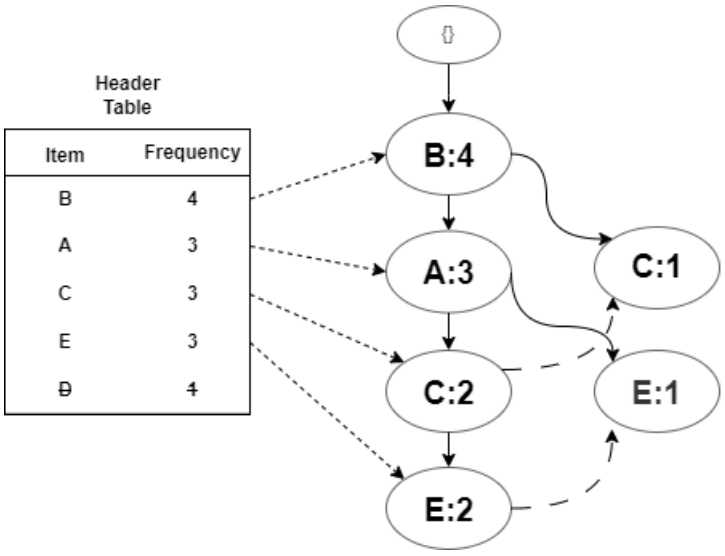

Figure 2.1: FP-tree construction, TBD scanned.

each item into the tree by their respective ordering of the tree's header table [HPY00], incrementing the equivalent nodes [2] support. For each item inserted into the tree, let us define the current number of inserted items of a transaction as k, the item is placed at tree depth[3] k+1 with the node's parent[4] matching the previously inserted element. During the insertion of new nodes, a chain of references is created. The header table stores a reference to the first inserted node of the equivalent item, and each node then stores a reference to the next, newly created node, of the equivalent item. A visual demonstration of FP-tree construction of Table 2.1 is provided in Figure 2.1. References are displayed through arrows.

[2] A **node** is a tree element consisting of five attributes: an item, a support value, a parent node, a list of child nodes, and a reference to another node in the tree.

[3] **Depth** refers to the current level in a tree. The topmost node is said to have a depth of 0, whilst the next lower level (e.g., child of the topmost node) has a depth of 1.

[4] A **parent** is a connected node of a lower depth.

Step 2: Mining the FP-tree

After the construction of an FP-tree, the FP-tree is mined to generate all frequent patterns. Let us denote the current tree t as our constructed FP-tree and the current itemset as i_c. For each itemset i_{ht} in the header table of t, perform $i_c \cup i_{ht}$ where support $= i_{ht}$ [HPY00]. Next, construct a conditional FP-tree, denoted as t_n, for $i_c \cup i_{ht}$ by creating a header table with each tree item, removing itemsets with total support $< minsup$ from the header table, and having each parent of $i_c \cup i_{ht}$ as nodes in this tree with a support of i_n. This process is repeated where $i_c = i_c \cup i_{ht}$, and $t = t_n$ until $t = \{\emptyset\}$ [HPY00]. The process of going through each item of the tree is repeated. A conditional tree of itemset X is denoted as an X-conditional tree. For ease of understanding a visual demonstration of this process from Table 2.1 is provided in Figures 2.2, 2.3, 2.4, and 2.5.

Optimization 1: Single Path Optimization

Single path optimization is an improvement to FP-growth by using combinations for trees that contain a single path[5]. Frequent patterns can be obtained through combinations. Consider the C-conditional in Figure 2.4. All combinations of the node along the path yields frequent patterns. In this case, $\{B\} : 3$, $\{A\} : 2$, yields $\{A, C\} : 2$, $\{B, C\} : 3$, and $\{A, B, C\} : 2$. Support of the combined itemset is the support of the minimum singular item. Computation time and memory improvements are made by skipping repeated conditional tree construction.

Example 2.1.2. *Let us suppose there exists an FP-tree provided in 2.1 with minsup =*

[5] A **path** is a set of connected nodes.

3, *and single path optimization is not applied. Scan through each item of the header table. Starting with* $\{B\}$*, the B-conditional tree is empty. Moving to header item* $\{A\}$*, the A-conditional tree* $\{B\} : 3$ *is created, item* $\{C\}$ *is ignored for falling below minsup* ($1 < minsup$)*.* $\{A, B\} : 3$ *is frequent. The AB-conditional tree is empty. Moving to header item* $\{C\}$*, the conditional tree* $\{B\} : 3$ *is created, with item* $\{A\}$ *and* $\{C\}$ *ignored for falling below the minsup* ($1 < 2 < minsup$)*.* $\{B, C\} : 3$ *is frequent. The BC-conditional is empty. Finally, moving to the header item* $\{E\}$*, the E-conditional tree contains* $\{B\} : 3 \rightarrow \{A\} : 3$*, with* $\{C\}$ *ignored for falling below minsup* ($2 < minsup$)*.* $\{B, E\} : 3$ *and* $\{A, E\} : 3$ *are frequent patterns. The BE-conditional tree is empty. The AE-conditional tree contains* $\{B\} : 3$*.* $\{A, B, E\} : 3$ *is frequent. The ABE-conditional tree is empty, and the program halts execution.*

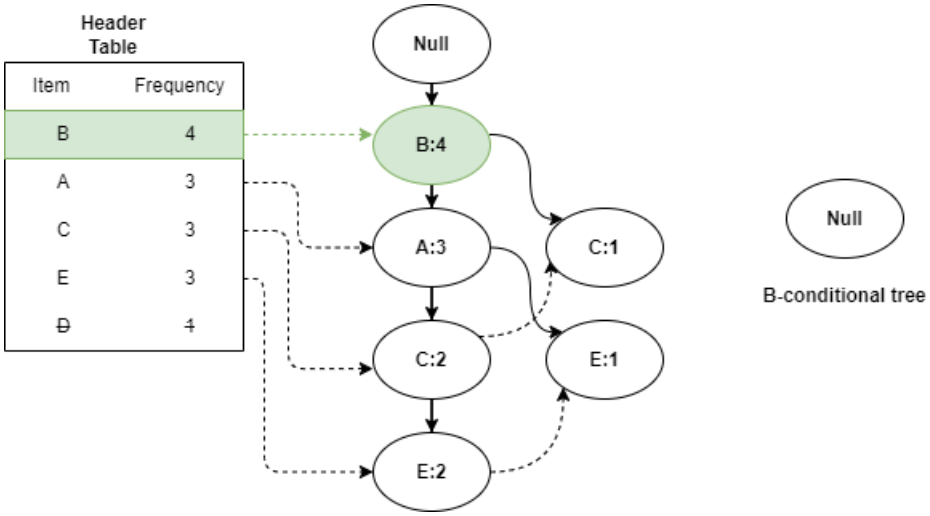

Figure 2.2: FP-growth following initial header item B. No single path optimization.

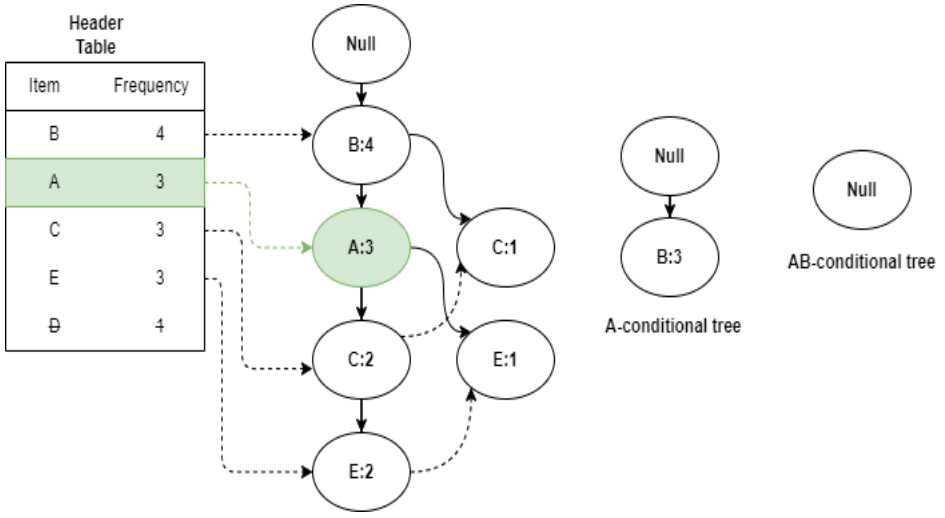

Figure 2.3: FP-growth following initial header item A. No single path optimization.

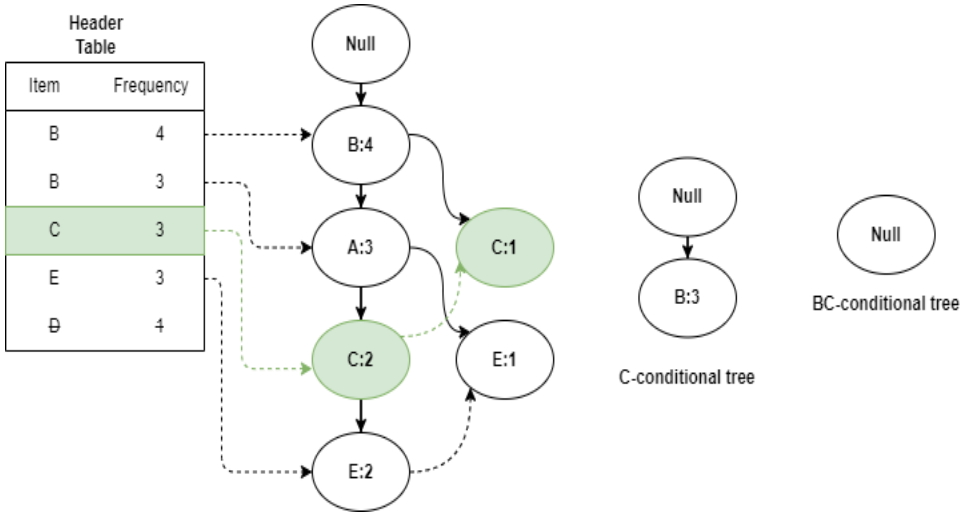

Figure 2.4: FP-growth following initial header item C. No single path optimization.

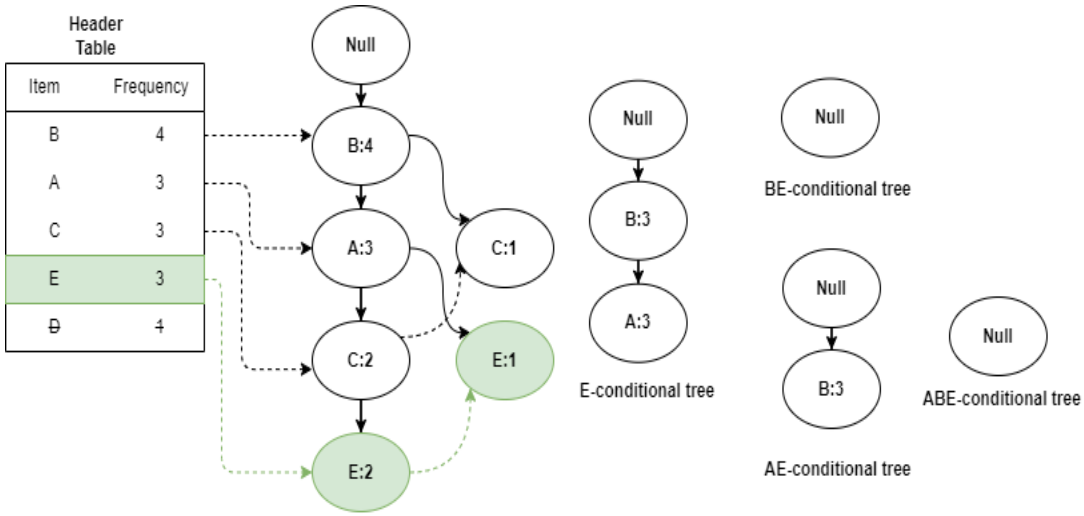

Figure 2.5: FP-growth following initial header item E. No single path optimization.

2.2 Exact Rare Pattern Mining

Other applications include determining outliers such as in cases of fraud [GC20], determining uncommon events that are unlikely to be performed by the user. Similarly, infrequent mining can be utilized in healthcare, where specific conditions are unlikely to occur among a large population. Associations and outlier detection can be useful for producing treatments or discovering and predicting disease [GC20]. This calls for the discovery of uncommonly occurring, however interesting patterns, of which, is that of *rare pattern mining*.

Definition 2.2.1. *Given an itemset X and a user-defined maximum support threshold (maxsup), an itemset X is defined to be a* **rare itemset** *if:*

$$support(X) < maxsup \tag{2.1}$$

Similar to frequent pattern mining, rare pattern mining typically discovers patterns through a transactional database. For future reference of Chapter 2.2, suppose there exists a transactional database given in Table 2.4. Whilst rare patterns are infrequent, it is not necessarily the case where all infrequent itemsets are of interest. Definitions of potentially interesting infrequent itemsets are listed below, which capture a subset of all infrequent itemsets.

2.2.1 Perfectly Sporadic Rule

Ko and Rountree [KR05] introduce the discovery of perfectly sporadic rules via the capturing of rare patterns. A perfectly sporadic rule is a logical implication such

Table 2.4: Transactional database 2.

Transaction ID	Items
0	A, B, C, E
1	A, C
2	B, C, E
3	A, C, D

that all subsets of itemsets are rare and their corresponding association maintains high confidence.

Definition 2.2.2. *An association rule $A \Rightarrow B$ is considered to be* **perfectly sporadic** *[KR05] if its confidence meets or exceeds a user-defined minimum confidence threshold ($minconf$):*

$$Confidence(A \Rightarrow B) \geq minconf \tag{2.2}$$

where $\forall x \subseteq (A \cup B)$, $support(x) < maxsup$

For future reference, the term perfectly sporadic itemset will be used as an itemset used to generate perfectly sporadic rules (i.e., all subsets are rare).

2.2.2 Rare-Item Itemset

A rare-item itemset ensures that there exists at least one item within an itemset to be rare Tsang *et al.,* [TKD11]. Given an itemset $X = \{x_0, x_1, ...x_n\}, n \geq 0$, this can be formally defined as:

$$\exists x \in X, Support(x) < maxsup \,\&\, Support(X) < maxsup \tag{2.3}$$

2.2.3 Apriori-Inverse

Apriori-inverse is a rare mining algorithm proposed in 2005 by Ko and Rountree used for extraction of perfectly sporadic rules [KR05]. As the name suggests, Apriori-inverse takes inspiration from the Apriori algorithm described in Section 2.1.1. The difference between Apriori and Apriori-inverse is with respect to the defined thresholds. In frequent pattern mining, Apriori discovers an itemset X such that, $support(X) \geq minsup$, for a user-defined minimum support threshold $minsup$. Apriori-inverse utilizes a maximum support threshold $maxsup$, and an absolute minimum threshold ($absmin$). That is, $absmin \leq Support(X) \leq maxsup$. Other aspects of this algorithm remain the same. Section 2.1.1 describes the initial scan, join step, prune step, and large step of the Apriori algorithm. The initial scan is adjusted to ensure $absmin$ and $maxsup$ thresholds are met as opposed to $minsup$. Similarly, the prune and large step is adjusted to ensure the $absmin$ threshold.

Example 2.2.1. *Let us suppose there are the thresholds, absmin = 1, maxsup = 4. Continuing the algorithm through Table 2.4 after the initial database scan with k=1;* $\{A\}$ *and* $\{B\}$ *generates* $\{A, B\}$. $\{A\}$ *and* $\{E\}$ *generated* $\{A, E\}$. $\{B\}$ *and* $\{E\}$ *generated* $\{B, E\}$. *A database scan is performed to determine support,* $\{A, B\}$ *and* $\{A, E\}$*fall below the absmin threshold and is thus too low of support to be meaningful. No size 3 itemsets are generated with* $\{B, E\}$*, and thus the algorithm stops. The process of this algorithm is visually demonstrated in Tables 2.5 and 2.6.*

Table 2.5: Size-1 itemsets from database scan.

C1: Initial Scan

Itemset	**Support**
C	4
A	3
B	2
E	2
D	1

L1: Trim Frequent and Uninteresting Items

Itemset	**Support**
A	3
B	2
E	2

Table 2.6: Remaining rare itemsets.

C2: Join and Prune Step

Itemset
A, B
A, E
B, E

L2: Trim Uninteresting

Itemset	**Support**
~~A, B~~	~~1~~
~~A, E~~	~~1~~
B, E	2

Resulting Rare Patterns = $L1 \cup L2$

2.2.4 RP-Growth

Rare Pattern-Growth (RP-Growth) is proposed by Tsang *et al.,* in 2011 [TKD11]. RP-Growth is an improvement to the Apriori-inverse algorithm [KR05] by no longer generating candidates or performing excess database scans, taking inspiration from the FP-growth algorithm described in Section 2.1.2 [HPY00]. Unlike Apriori-inverse, the proposal of RP-Growth seeks to discover the complete set of rare-item itemsets, as opposed to all perfectly sporadic itemsets. A rare-item itemset is an itemset that contains at least one rare item. Similar to that of FP-trees, for any path along a tree, a parent node will have a support greater than or equal to that of its descendants. The construction of the tree ensures that a conditional tree of a non-rare itemset prefix will not be a rare-item itemset [TKD11]. RP-growth thus works in the same fashion as FP-growth. Tree construction and initial mining of the tree are adjusted to match the definition of rare-item itemsets.

Example 2.2.2. *Let us suppose there are the thresholds absmin = 1, maxsup = 4. Continuing the algorithm through Table 2.4 after the support count of each size-1 itemset is determined; rare itemsets and their corresponding transactions are stored for future reference. The transactions associated with an itemset X can be denoted as* X_T*, where* $X = \{T_0, T_1, ..., T_N\} N \geq 0$*. The resulting rare items are as follows,* $\{A\}$*:3* $A_T = \{T_0, T_1, T_3\}$*,* $\{B\}$*:2* $B_T = \{T_0, T_2\}$*,* $\{E\}$*:2* $E_T = \{T_0, T_2\}$*. The tree is constructed similarly to the FP-tree for all transactions:* $A_T \cup B_T \cup E_T$ *as shown in Figure 2.6. Rare items are denoted as a circle while frequent items are denoted as a square. Since the algorithm runs through only rare items in the tree, the header table in FP-growth contains only rare items from the initial tree construction. For*

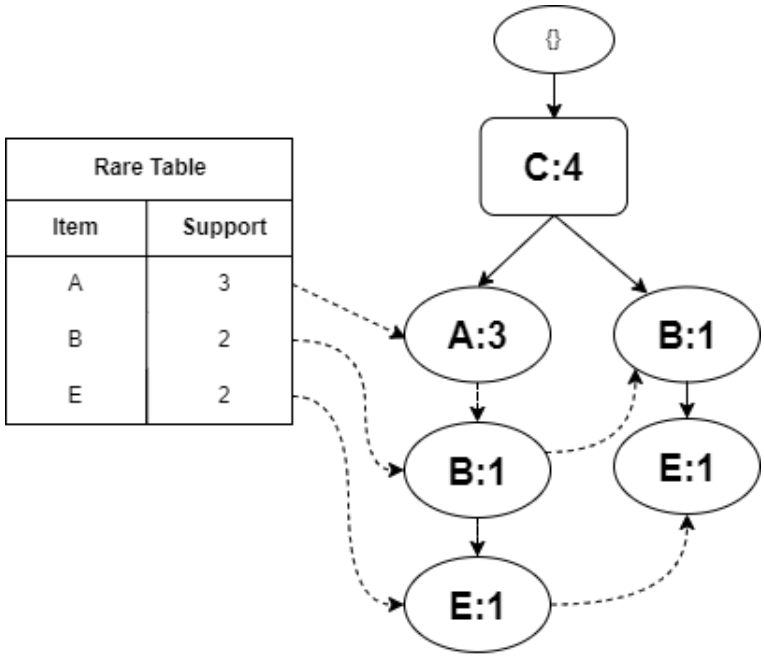

Figure 2.6: RP-Tree construction. $absmin = 1$, $maxsup = 4$.

each item in this table, the conditional patterns and conditional tree is generated. The algorithm then calls FP-growth for the resulting conditional tree [TKD11]. See section 2.1.2 on the FP-growth algorithm.

2.3 Mining in Noisy Data

In the real world, datasets often contain noise. This influences exact matching techniques in data mining. To resolve this issue, measures of discovering patterns in noisy data apply lenience to the definition of a frequent itemset. That is, in the case of frequent pattern mining, support no longer requires the number of occurrences of the whole itemset to be greater than or equal to *minsup* to be classified as frequent. Various definitions to allow this lenience have since been proposed, of which, a variety

of proposed techniques are discussed.

Table 2.7: Bitwise[6]transactional database 3.

Transaction ID	A	B	C	D	E
0	1	1	0	1	0
1	1	1	0	1	0
2	1	0	0	1	0
3	1	1	0	0	0
4	1	1	0	1	0
5	1	0	1	1	0

2.3.1 Fault-Tolerance

In 2001, Pei *et al.,* [PTH01] propose fault-tolerant (FT) itemsets as a solution to discovering itemsets with missing information in a dataset. The following definitions are introduced in the newly proposed model.

Definition 2.3.1 (Fault-Tolerance Factor)**.** *A user-define* **fault-tolerance threshold** *that determines the margin of eligible error in counting the support of an itemset. Denoted as* $\delta > 0 \in \mathbb{Z}$.

Definition 2.3.2 (FT-Contained itemset)**.** *An itemset* X *is* **FT-contained** *[PTH01] in a transaction* $T = (id, X)$ *if* $|X \cap X| \geq (|X| - \delta)$*. In other words, a transaction must contain a sufficient number of items of* X.

Definition 2.3.3. *Given a fault tolerance* δ *and minimum item quantity* min_sup^{item} *as user-defined variables, an itemset* X *is an* **FT-frequent itemset** *if and only if*

[6]For a given transaction (i.e., row), an item (i.e., column) exists if it is a 1, whilst does not exist if there is a 0.

the following conditions apply [PTH01]:

$$s\bar{u}p(X) \geq min\_sup^{FT} \tag{2.4}$$

$$\forall x \in X, s\bar{u}p_{B(X)}(x) \geq min\_sup^{item} \tag{2.5}$$

where $s\bar{u}p$ is defined as the number of FT-containing transactions X contains, with FT-containing being a transaction containing at-least $|X| - \delta$ items of X. min_sup^{item} denotes the minimum $s\bar{u}p$ for all items within the itemset X, and B(X) are all transactions FT-containing X.

The above definitions apply lenience to the traditional definition of support threshold. Consider $\delta = 1, min\_sup^{FT} = 2, min\_sup^{item} = 1$, in Table 2.7 itemset a,b,c is then frequent. Traditionally with frequent pattern mining, with $minsup = 2$, $\{a, b, c\}$ is infrequent. As a certain number of items appear within a transaction, and the items individually occur frequently enough within these transactions, then the transaction can contribute to the support count of the pattern, and thus, the pattern may be discovered.

2.3.2 Approximate Itemsets

In 2006 Liu *et al.,* propose an approximate model that includes both a row and column error tolerance to a bitwise transactional database [LPS+06], in addition to providing a tolerance that is proportional to an itemsets length. Three thresholds are used, $\epsilon_r, \epsilon_c \in [0, 1], minsup > 0$. Unlike FT-frequent itemsets, this approximation model recovers *core patterns* in the presence of noise. A core pattern is one in which

the exact support is larger than a defined *absmin* threshold.

Definition 2.3.4 (Approximate Frequent Itemset (AFI))**.** *An itemset is a frequent AFI if the following conditions hold [LPS⁺06];*

1. $\forall i \in FT(X), \frac{1}{|X|} \sum_{j \in X} D(i,j) \geq (1 - \epsilon_r)$
2. $\forall j \in X,$
   $\frac{1}{|FT(X)|} \sum_{i \in FT(X)} D(i,j) \geq (1 - \epsilon_c)$
3. $|FT(x)| \geq s|T|$

Rules 1 and 2 enforce row error and column error thresholds, respectively. Rule 3 enforces minimum support threshold criteria. Additionally, as the itemset is a core pattern, $Support(X) > absmin$.

Consider the transactional database in Table 2.7. With traditional mining of $minsup = 3$, $\{d\}$ yields a frequent itemset, whilst $\{c, d\}$ and $\{c\}$ are infrequent. In the case of AFI's, with $\epsilon_c = \frac{1}{5}$, $\epsilon_r = 0.5$, $\{d\}$ and $\{c, d\}$ yield a frequent itemset, whilst $\{c\}$ remains infrequent. With exact mining, the anti-monotone property enables effective pruning by excluding all supersets of infrequent itemsets. In the case of approximate mining, there exists a possibility of an infrequent itemsets superset remaining frequent. As such, the success of the anti-monotone property that led to the Apriori and FP-growth algorithms is inapplicable. Liu *et al.,* introduce an alternative measure of support to allow pruning without the anti-monotone property [LPS⁺06]:

$$minsup^k = max\left(0, minsup \times \left(1 - \frac{k\epsilon_c}{\lfloor k\epsilon_r \rfloor + 1}\right)\right) \tag{2.6}$$

where k denotes the length of an itemset, and $minsup$, ϵ_r, and ϵ_c are our user-defined thresholds. When the support of an itemset is smaller than the defined $minsup^k$, then the itemset is removed from consideration as a potential AFI itemset. Mining algorithms that discover AFI's utilizes this pruning strategy as a substitute for the well-known anti-monotone property.

Another measure of anti-monotone property substitution was presented in 2006, where Liu *et al.,* . proposed the AFI mining algorithm [LPS+06]. The AFI mining algorithm uses an Apriori-like approach by using similar techniques to the initial database scan, generation, and determination of frequent itemsets described in Section 2.1.1. Since the anti-monotone property is inapplicable, alternative pruning approaches are required in the generation step of an Apriori algorithm to avoid generating all combinations of itemsets. One such pruning method is that of 1/0 extensions. Precise definition of 1/0 extensions is provided below [LPS+06];

1-Extension if $\lfloor (k+1)\epsilon_r \rfloor = \lfloor k\epsilon_r \rfloor$, then an infrequent size k itemset will yield an infrequent size k+1 itemset;

0-Extension if $\lfloor (k+1)\epsilon_r \rfloor = \lfloor k\epsilon_r \rfloor + 1$, then a frequent size k itemset will yield a frequent size k+1 itemset;

Consider an itemset of length k, and row tolerance ϵ_r. With k=2 $\epsilon_r = 0.5$, $\lfloor (2+1)0.5 \rfloor = 1 = \lfloor 2 * 0.5 \rfloor$. A 0 extension occurs and so an additional 0 is tolerated. In other words, any size-2 AFI may yield a size-3 AFI. No pruning is performed under a 0 extension. For k = 3, $\lfloor (3+1)0.5 \rfloor = 2 = \lfloor 3 * 0.5 \rfloor + 1$. A 1-extension occurs, implying no additional 0s are tolerated. Any size-2 itemset that is not an AFI will not yield a size-3 AFI. 1-extensions allow for superset pruning. AFI mining algorithm

utilizes 1/0 extensions as an alternative pruning measure to the Apriori algorithm. Note that 1/0 extensions prune by ϵ_r and k. Additional pruning measures by ϵ_c may take place. For an itemset X, any item $i \in X$, the following must be satisfied [CYH08];

$$\exists i \in k, \frac{sup(i)}{sup(X)} \geq (1 - \epsilon_c)$$

Intuitively, this property makes sense. Let us suppose $\epsilon_c = 0.5$. If one item within an itemset maintains less than half the support of the itemset itself, then ϵ_c fails. ϵ_c describes the fraction of required support for each individual item within an itemset.

2.3.3 Apx Pattern-Growth

More recent work in the area of AFI mining is that of Bashir and Lai who introduced a tree-based approach to discovering AFI's in 2021 [BL21]. The tree-based solution first creates an FP-tree in the same fashion as the FP-growth algorithm introduced in Section 2.1.2. The FP-tree is then mined for AFI itemsets.

There are three new structures used to mine the FP-tree. An approximate-conditional pattern (apx-pattern in short) contains four key components for an itemset [BL21]:

1. A list of potential superset items
2. Patterns support value
3. An error tolerance
4. Frequencies for each item of the captured itemset

Each attribute of an apx-pattern will be denoted as I:$\langle(1),(2),\epsilon_r=(3),(4)\rangle$, where I is the underlying item used to create the given apx-pattern, and (number) represents the key components numerical value listed above. For simplicity, an apx-pattern will be referred to as an apx-pattern.

The second additional structure is an apx-FP-tree. Similar to the well-known FP-tree, except with the final additional structure as leaf nodes connected by a list [BL21]. The final additional structure is the apx-CP-Table, which, maintains (2), (3), and (4) of an apx-pattern and is treated as a leaf node in an apx-FP-tree. This algorithm differs from the FP-growth in two key areas. Firstly, unlike the traditional FP-growth, all possible size-2 combinations of frequent size-1 itemsets are tested as an AFI itemset [BL21]. Let us consider $\epsilon_r = 0.5$, then 50% of an itemset is required to contribute to its underlying support. Each item of an itemset has the potential to contribute to the support of a superset. For example, consider the itemset $\{A, C\}$ in Table 2.7. $\{A\}$ occurs with the item $\{C\}$ without the item $\{B\}$ in transaction 5, and so with $\epsilon_r = \frac{1}{3}$, transaction 5 approximately supports $\{A, B, D\}$. With the traditional FP-growth algorithm, this contribution of support would not be measurable. Instead, apx-FP-growth uses apx-patterns for each item of an itemset. Apx-patterns are then used to generate an apx-FP-tree and mined to discover supersets of the given itemset.

Mining FP-tree $\{A, B\}$ Example

Let us consider the itemset $\{A, B\}$ without single-path optimization where $\epsilon_c = \frac{2}{6}$, $\epsilon_r = \frac{2}{3}$, $absmin = 1$, and $minsup_{apx} = 2$. Suppose an FP-tree is constructed in the same form as the FP-pattern growth algorithm as shown in Figure 2.7.

First conditional patterns of item A and B are independently found: $A : \langle\{\}\rangle$:6,

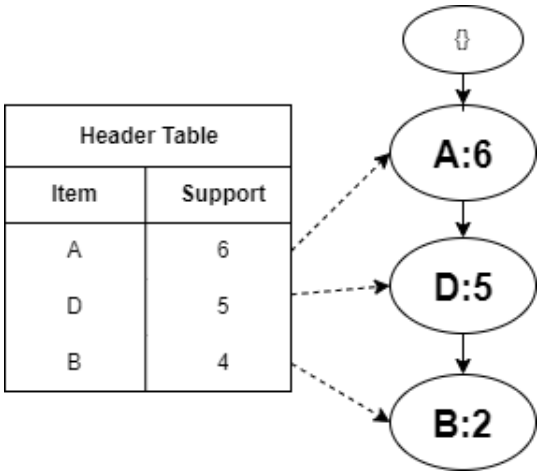

Figure 2.7: FP-tree constructed from Table 2.7

$B : \langle\{ADB\}\rangle$:2. Next, apx-patterns are formed: $A : \langle(), sup : 4, \epsilon_r = 50\%, A : 4\rangle$, $B : \langle\{D\}, sup : 2, \epsilon_r = 100\%, A : 2\, B : 2\rangle$. Patterns of A are a subset of B, and thus the supports are subtracted. $absmin < minsup_{apx} < 6$. All itemsets are present at least $\frac{2}{6}$ times, and $\frac{2}{6} = \epsilon_c$. Therefore, $\{A, B\}$ is an AFI. Finally, an apx-FP-tree is constructed. Itemset $\{D : 2\}$ will remain as the only existing item in the header table of the apx-FP-tree. $\{A, B\} \cup \{D\}$ is performed to yield the itemset $\{A, B, D\}$. D's conditional patterns of $\{AD\} : 5$ are formed and converted to the apx-patterns: $\langle\{\}, sup : 5, \epsilon_r = \frac{2}{3}, A : 5\, D : 5\rangle$. A's apx-pattern is a subset of D in addition to D being a subset of B; supports are adjusted accordingly. After converting B's conditional patterns to match the itemset $\{A, B, D\}$, the following apx-patterns are formed: $D : \langle\{\}, sup : 3, \epsilon_r = \frac{2}{3}, A : 3\, B : 0\, D : 3\rangle$, $B : \langle\{\}, sup : 2, \epsilon_r = 100\%, A : 2\, B : 2\, D : 2\rangle$. $\frac{2}{3} \geq \epsilon_r \,\&\, 1.0 > \epsilon_r$, therefore there are 5 approximately supporting transactions of $\{A, B, D\}$. $absmin < minsup_{apx} < 5$. All itemsets are present at least $\frac{2}{5}$ times, and $\frac{2}{5} > \epsilon_c$. Therefore, $\{A, B, D\}$ is an AFI. An apx-FP-tree is constructed, which, yields no further itemsets so the algorithm halts execution.

2.4 Summary

In this chapter, related works are reviewed. A common technique for discovering interesting patterns includes generation approaches (e.g., Apriori), or tree-based approaches that are used to reduce database scans (e.g., FP-growth). Both generation and tree-based approaches have been applied in frequent, rare, and noise pattern mining. In the case of rare pattern mining, computational constraints occur as a result of the *rare itemset problem*. To absolve this issue, interesting rare itemsets are defined more precisely (e.g., perfectly sporadic rule, or rare-item itemset). Lastly, whilst frequent and rare mining techniques are commonly used in practice, they are insufficient under the presence of noise. That is, when noise is present in the dataset, information is lost. As a result, models have been proposed (e.g., approximate frequent itemset) in order to recover the true set of patterns in addition to ignoring the false set of patterns that are created from the presence of noise when *exact* mining is used.

Chapter 3

Rare Itemsets in Noisy Data

In Section 2.2, the problem of rare mining is discussed, in addition to the discovery of itemsets under the presence of noise in Section 2.3. While these problems spark interest in research, to the best of my knowledge, no such work considers the application of both problems. It is not always the case where common items are interesting. For instance, suppose a retail store wishes to organize store layout based on the relationships of how items are purchased together; to maximize customer convenience. Not all products are commonly purchased, so, retrieving a convenient layout across all customers requires finding rare associations. This work considers the case where data is under the influence of noise as a result of information loss whilst uncommon patterns are of interest. In the case of approximate frequent patterns, noise may lead to a frequent itemset becoming infrequent. As a result, frequent patterns may be lost entirely. With rare patterns, noise is less likely to cause patterns to be lost since a pattern's support must fall below the *absmin*; which is typically set to a minimum measure of significance. I argue rare mining in noisy data is still of interest for two

reasons. **1)** Rare patterns may be lost when a minimum support is still enforced. **2)** Frequent patterns lost as a result of noise are not of interest in rare pattern mining. **3)** Metrics such as confidence are used to evaluate the strength of an association; such metrics may no longer capture interesting associations. Recall, if there exists a rule $\{A\} \implies \{B\}$, confidence is calculated as $\frac{Sup(AUB)}{Sup(A)}$.

Suppose data is noisy. From the calculation of confidence, noise can result in a loss of $Sup(A)$ without any loss to $Sup(AUB)$ from the loss of item A. As a result, the confidence under noise is *increased* when compared to the true confidence if the data were not noisy. The reverse may hold. Suppose noise resulted in a loss of $Sup(AUB)$, but retained $Sup(A)$ from a loss of the item B. In this case, the confidence under noise is *decreased* when compared to the true confidence. Noise causes the risk for patterns to be misclassified by metrics calculated through support. When applying noise to each item in a retail dataset [1], a significant loss of interesting rules under increased levels of noise is observed. Figure 3.1 presents the loss of interesting sporadic rules when an amount of noise probability of an item being removed is added to the dataset. For instance, with $minconf = 95\%$, a 9% decrease in interesting associations with 3% item removal probability can be observed from the tested dataset.

[1] Properties of this retail dataset are provided in Chapter 4.

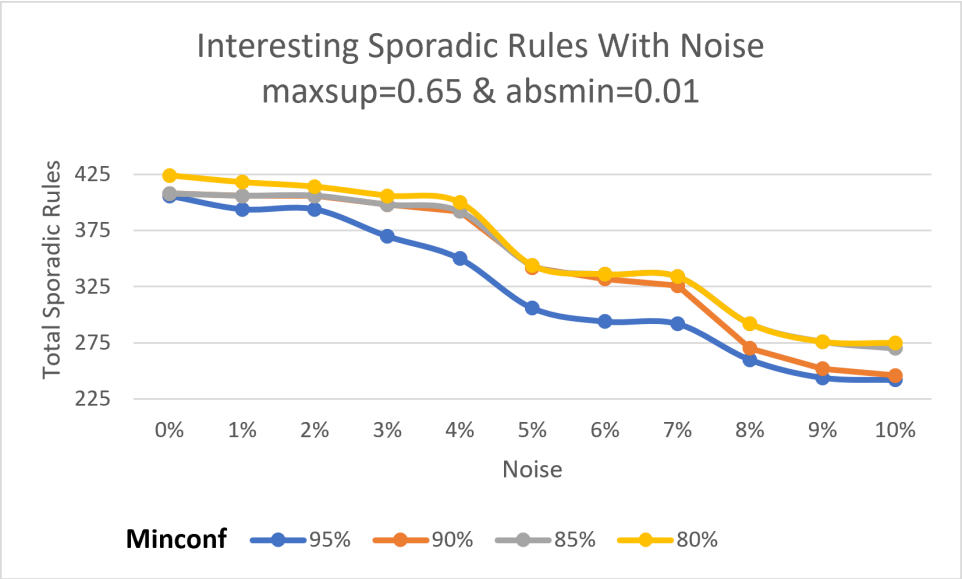

Figure 3.1: Sporadic rules in noisy *Retail* data

In the proposal of Apriori-inverse by Ko *et al.*, perfectly sporadic itemsets are deemed as interesting and are captured by the algorithm [KR05]. The severity of the computation required to discover all rare itemsets is too large, so, more precise definitions of rare have been defined. In order to discover every rare item set, one must reduce the minimum support threshold in traditional frequent pattern mining to a sufficiently low amount, discovering only those patterns that do not exceed the minimum support threshold. This in effect significantly reduces run time and generates redundant rules that are uninteresting; as by the *rare itemset problem* [KR05]. Various definitions of interesting rare itemsets such as sporadic, minimally rare, or rare-item, attempt to resolve the *rare item problem* by ignoring those that may otherwise be deemed uninteresting.

Rare in Noise

Approximate mining is computationally more expensive due to failure of the *anti-monotone property* (see section 2.1 on the property in traditional datasets). Under the definition of AFI, given $\epsilon_r = 0.6$, $\epsilon_c = 1.0$ a dataset $D = [[A, B, C], [A, C]]$ yields approximate support (Sup_{apx}) $Sup_{apx}(\{A, B\}) = 1, and\, Sup_{apx}(\{A, B, C\}) = 2$. Thus, it is not necessarily the case where subsets support is less than or equal to that of any superset. Under the presence of noise however, rare itemsets may be incorrectly deemed as rare (*i.e., false positive*), or incorrectly deemed as not rare (*i.e., false negative*). The problem of discovering rare itemsets in the presence of noise is introduced. This problem is further expanded through an adjustment of the definition of rare-item itemsets utilized by the RP-growth algorithm [TKD11].

3.1 Problem Statement

I seek to answer the first question in Section 1.2.1, *How can rare itemsets be discovered?*. In order to answer this question, a model to describe an interesting itemset is defined, and then introduce an approach to discovering such itemsets.

A trivial definition of a rare itemset is one that is infrequent. As support lenience is being applied, a definition for rare itemsets in noisy data can be formulated through modifications to that of an approximate frequent mining approach. For a transactional database, extraction of frequent itemsets can be described with three thresholds ϵ_c, ϵ_r, $maxsup \in [0, 1]$. Following the formal definition by Cheng et al [CYH08], Let a transactional database be denoted as D with D(i,j) having a bitwise tabular value at transaction i item j, for itemset X, and a set of transactions $FT(X)$

that approximately support X.

The definition of AFI is reverted to define an approximately rare itemset (ARI). See Definition 3.1.1.

Definition 3.1.1 (Rare in Noise)**.** *An itemset is* **rare** *if:*

$$\frac{|FT(X)|}{|D|} < maxsup$$

and the following two conditions hold where $|FT(X)| > 0$*;*

$$\forall i \in FT(X), \frac{1}{|X|} \sum_{j \in X} D(i,j) \geq (1 - \epsilon_r)$$
$$\forall j \in X, \frac{1}{|FT(X)|} \sum_{i \in FT(X)} D(i,j) \geq (1 - \epsilon_c)$$

In the case of noise, ϵ_c filters itemsets with an item that does not occur often in approximately supported transactions, whilst ϵ_r provides lenience to support. As a result, the set of patterns that truly exist if the noise were to be absent is approximated. If the support of an itemset is too small, however, its existence will not be measurable. In the case of mining AFI itemsets, *core patterns* are recovered [LPS$^+$06]; of which, have absolute support larger than a defined *absmin* threshold. The usage of the *absmin* must be adopted in the case of mining rare itemsets in the presence of noise to retain a *core pattern* recovery model.

While the above definition is an intuitive description of approximating rare patterns, it suffers from the **rare itemset problem**. That is, capturing the complete set of rare patterns is computationally expensive, and often captures ones which are uninteresting. An approximation is thus applied to a type of rare itemset, more specifically, to that of rare-item itemsets described in Section 2.2.2. Rare-item itemsets are both applicable to tree-based approaches, such as RP-growth [TKD11] and

additionally, have measured capturing of at least 92% of useful yet rare rules with a confidence metric [TKD11].

Definition 3.1.2 (Approximate Rare-Item Itemset)**.** *An itemset is a rare-item itemset if there exists at least a rare item within the itemset, and the itemset itself is rare. In order to accommodate for noise, these thresholds are appended to the original definition of rare-item itemset. That is, an itemset X is an approximately rare-item itemset if;*

$$\exists x \in X, Support(x) < maxsup$$

$$Support_{apx}(X) < maxsup$$

$$\forall x \in X, \sum_{i \in FT(X)} D(i,x) \geq Support_{apx}(X) * \epsilon_c$$

In this case, any size-1 itemset X whose support is greater than or equal to $maxsup$ will be ignored. Only itemsets that contain at least one size-1 itemset less than $maxsup$ are captured. Size-1 itemsets are not approximated by the ARI model. Therefore, the accuracy of size-1 itemsets is relied on to that of *core patterns*. As such, if an itemset X occurs at most $absmin$ times (i.e., $Sup(X) \leq absmin$) where $absmin \geq 0$ and $absmin < maxsup$, then the itemset has too low of support to be classified as interesting due to the presence of noise. While the $maxsup$ threshold provides a boundary for rare itemsets in noise, it does not accommodate pattern recovery in the case of information in a dataset being lost. If the true support of a pattern is decreased as a result of noise, the minimum $absmin$ threshold to identify core patterns is further relied on.

In rare pattern mining, it is commonplace to use a similar lower bound threshold to ignore patterns [TKD11] that have too low of a frequency to be interesting for

a researcher. Similarly, a support too low is highly sensitive to noise. That is, added information may be deemed as interesting leading to *false positives*, or lost information may not be captured leading to *false negatives*. In order to capture patterns that maintain interest from their support, a *minsup* threshold is applied to approximate rare-item itemsets, such that $absmin \leq minsup$. An itemset X with $Sup_{apx}(X) \geq minsup$, $Sup(X) > absmin$ is required to be classified as interesting, in addition to meeting the rare-item itemset requirements specified in Definition 3.1.2. In this regard, *absmin* denotes the minimum allowed *exact* tolerance to capture core patterns, whilst *minsup* and *maxsup* threshold denote a range of the approximate support of interest.

For the remainder of this chapter, suppose there exists a transactional database provided in Table 3.1. Two algorithms for discovering approximately rare-item itemsets with the use of core patterns defined by *absmin* whilst maintaining a lower bound with *minsup* are introduced.

Table 3.1: Transactional database 4.

Transaction ID	A	B	C	D	E
0	1	0	1	0	1
1	1	1	0	1	0
2	0	1	1	0	0
3	1	0	0	0	0
4	1	1	0	1	0
5	0	1	1	0	0

3.2 Naive Algorithm

A Naive approach is the simplest for the discovery of all approximate rare-item itemsets. The algorithm begins with an initial database scan by counting the support of each size-1 itemset. The remainder of the algorithm consists of two primary stages. Generation of each size-k itemset, and a database scan to count the approximate support of each itemset.

Generation of size-k itemsets is as follows; for any ARI itemset of length k, the generation of a possible length k + 1 itemset is given with a join of all size-1 itemsets. For example, in Table 3.1 size-1 itemsets are $\{A\}$, $\{B\}$, $\{C\}$, $\{D\}$, $\{E\}$. Size-2 itemsets from this join are as follows: $\{A, B\}$, $\{A, C\}$, $\{A, D\}$, $\{A, E\}$, $\{B, C\}$, $\{B, D\}$, $\{B, E\}$, $\{C, D\}$, $\{C, E\}$, $\{D, E\}$. The process continues by appending each size-1 itemset to each other size-k ARI itemset. When size-k itemsets are generated, the approximate support of each itemset is counted through a database scan. Any transaction that contains at least $|X| * \epsilon_r$ items of X approximately supports X, and each item contributing to this support is accounted for. After the generation of each itemset, all thresholds are checked for ARI criteria. This process is an intuitive and simple solution.

3.3 ARI Pattern Growth

While a Naive approach provides a solution to the problem statement, they are often not useful in the real world. In this case, all possible ARIs are considered, and the database is scanned for all available sizes of itemsets to calculate approximate sup-

port. Therefore, a more practical pattern-growth [2] approach is proposed. Given row and column error thresholds ϵ_r, & $\epsilon_c > 0$, a maximum support threshold $maxsup > 0$, minimum support threshold $minsup < maxsup$, and an absolute minimum threshold $0 \leq absmin \leq minsup$, the ARI-growth approach discovers rare itemsets. This algorithm is inspired by the apx-pattern growth that discovers AFI itemsets, proposed by Bashir et al in 2021 [BL21]. Aside from discovering rare itemsets, there are a number of improvements in the proposed algorithm. ARI-growth saves excess computation by constructing only one tree and does not construct additional tables from an itemsets apx-conditional patterns in the AFI-pattern growth [BL21]. The proposed ARI-growth discovers all itemsets by recursively scanning apx-conditional patterns. Each apx-conditional pattern stores a reference to a node in the original RP-tree, which scans the tree to subtract a subset patterns support to accommodate for redundant support counting; in contrast to the AFI pattern growth where the support of supersets is subtracted from one another [BL21]. However, comparisons of supersets from apx-conditional patterns are still applicable. Additionally, as a result of the support requiring being larger than the *absmin*, and $absmin \geq 0$, an itemset must be fully contained within a transaction at least once. So, ARI-growth does not compute all combinations of potentially interesting size-2 itemsets from size-1 itemsets that are performed in the apx-pattern growth algorithm [BL21].

[2]Pattern growth is in reference to the mining aspects of the FP-pattern growth algorithm explained in Section 2.1.2.

Initial Tree Construction

This algorithm begins with a pre-processing step. First, the database is scanned and the initial support of all size-1 itemsets is counted. Itemsets with $Support(X) \leq$ *absmin* are ignored. The algorithm then orders each item by its frequency ordering and maps the corresponding order lexicographically. For ease of interpretation, Table 3.1 is pre-mapped with items A, B, C, and D having a respectively ordered support. Next, an RP-tree is constructed in the same form as the RP-growth. A 2nd database scan is performed to construct the RP-tree with its corresponding header table as described in Section 2.2.4, where items are accounted for by their mapped values. The initial RP-tree, and all thresholds are required for the remaining execution of this algorithm. Figure 3.2 denotes the resulting tree when applied to the dataset in Table 3.1 with $absmin = 1$ and $maxsup = 6$.

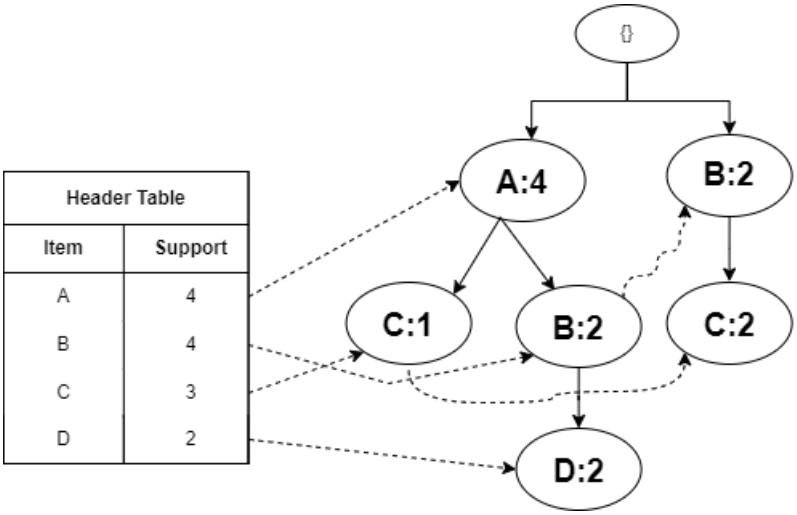

Figure 3.2: Initial RP-tree.

Apx-Conditional Pattern

An Approximate-conditional (apx-conditional) pattern is used in the pattern growth

approach that discovers AFI's [BL21]. This technique will be utilized to uncover ARI itemsets. In FP-growth, conditional patterns are a set of items that follow along the path of the tree from a given node to the tree root. In the case of approximate mining, additional information, such as the portion of existing items in the itemset, is required to determine approximate support.

Apx-conditional patterns in the AFI pattern growth algorithm [BL21] are amended. In this work, an apx-conditional pattern contains *six* key components, as described below;

1. An underlying item of X.
2. A set of superset containing items. That is, the items along the tree path that are excluded from the itemset captured by this pattern.
3. A Support count.
4. A row error tolerance.
5. A counter of each item for the itemset captured by this pattern.
6. A pointer to an RP-tree node.

Apx-conditional patterns are used to generate supersets and determine if an itemset meets all required thresholds. An apx-conditional pattern is denoted in the following manner, item I: $\langle$ {superset items}, sup:N, $\epsilon_r = R\%$, C:X $\rangle$, where I is an item of X, N is the number of supporting transactions, R is the row error, C:x is the support contribution of each item $C \in x$ with the support x, respectively following key component ordering 1 through 5. The final component of the apx-conditional

pattern is excluded from textual representation. For future reference, an individual pattern of apx-conditional patterns will be referred to as an apx-pattern.

Mining the RP-tree

Mining the initial RP-tree can be summarized in 4 steps. First, the algorithm begins by iterating through each rare size-1 itemset in the initial trees header table. Conditional patterns from rare size-1 itemsets are formed. Conditional patterns are then converted to apx-patterns. Rare size-1 itemsets are printed, and the algorithm continues recursively with apx-patterns of the itemset. Let us suppose the header table of an apx-pattern is given as $Y = i_0, i_1, ..., i_k, k \geq 0$. The recursive step for the previous apx-pattern works as follows: For each item in the header that satisfies *absmin*, Y_i, $X_n = X \cup Y_i$ is performed. A new conditional pattern is formed with the item Y_i, which is then inserted into the apx-patterns of X, a process in this thesis referred to as *upwards propagation* is performed. Next, X is checked for ARI criteria. If $Sup_{apx}(X) \geq maxsup$ or $Sup(X) \leq absmin$ or $Sup_{apx}(X) \leq minsup$ then X is not an ARI. ϵ_c is checked by ensuring all items i of X satisfy: $\frac{approximately\ supported\ transactions\ of\ i}{approximately\ supported\ transactions of\ X} \geq \epsilon_c$. This process continues when X_n satisfies *absmin* criteria. A pseudo-code of this algorithm is provided in Algorithm 1. The process of conditional pattern construction, conversion to apx-patterns, insertion, and upwards propagation is explained in detail below.

Conditional pattern construction

Let us suppose there exists a size-1 itemset $X = \{X_0\}$. Conditional patterns are generated by scanning the connected node of X_0 in the header table of the RP-tree, and for each node, take the item within the node i along the path to the root and

perform $i \cup X_0$. The support of the conditional pattern is equal to the support of the node containing X_0. This process is repeated for each connected node that contains X_0. For each conditional pattern, connected nodes of X_0 are stored.

Conditional pattern to apx-conditional pattern conversion

Suppose conditional patterns are converted to apx-conditionals for the itemset X. All six components of an apx-pattern are created. (1) is assigned to the underlying item used to discover the conditional patterns. (2) denote the set of items in the conditional pattern as T, and the set of supersets containing items is given as $T - X$. (3) is assigned the support of the conditional pattern. (4) row error is calculated as $\frac{|items \in X|}{|X|}$. (5) Each item of X is removed from T. Counters are set to the patterns supported in (3). Finally, (6) is assigned the pointer to the connected node from the conditional pattern. Using components (2), (3), and (4) of each apx-pattern, a header table is formed by summing the support (i.e., *component*(3)) of each fully supported approximate pattern's supersets (i.e., *component* (4) $\epsilon_r = 100\%$, for all items in component (3)).

Apx-pattern insertion

The following process occurs to convert X apx-patterns to X_n with the insertion of item Y_i. First, *upwards propagation* is performed. Next, Y_i is removed from component (2) of each apx-pattern. Y_i is inserted into the component (5) with its counter equal to the support of the pattern; row tolerance is updated accordingly. Finally, all items i in component (2) where $Y_i > i$ are removed. For example, let us suppose there exists an itemset $X = \{D, E, F\}$ with component (2) of any given apx-pattern containing any item in the itemset $\{A, B, C\}$. Let us further suppose

$\{B\} \cup \{D, E, F\}$ is performed by inserting B to apx-conditional of itemset $\{D, E, F\}$. In this case, items B and C are removed from component (2) of all apx-patterns for itemset $\{B, D, E, F\}$. The removal of component (2) items is an optimization that prevents excess recursive steps in the algorithm. $\{B, D, E, F\}$ will only consider the superset A to generate $\{A, B, D, E, F\}$, while the alternative superset $\{B, C, D, E, F\}$ will instead be generated by itemset $\{C, D, E, F\}$. Additionally, any pattern where component (2) is the empty set, and the row error tolerance is less than ϵ_r is ignored. That is, the pattern will never contribute to the support of any superset, nor result in redundant support counting by upwards propagation.

Upwards Propagation

Let X be an itemset $\{X_0, X_1, ..., X_n\}, n > 1$, and let $X_t \in X$ denote the underlying item that was previously converted to apx-patterns. Each apx-pattern is separated by component (1) by items of X that succeed X_t. For example, let n = 3 and t = 1, then X_2 succeeds X_t. Next, for each succeeding apx-pattern (e.g., apx-patterns with X_2 as component (1)) that had also contained X_t in component (2), propagate up the RP-tree from component (6) to subtract the support of a linked pattern to a node containing X_t. If any node item less than X_t is found, propagation ends. To optimize this process, when propagation ends, component (6) of the apx-pattern that began traversing the tree upwards is replaced to reference the final visited node. This optimization ensures the next stage of upwards propagation uses fewer steps when propagating up the tree. Upwards propagation ensures redundant support counting does not occur.

Algorithm 1 ARI-Growth

Require: ϵ_r, ϵ_c, $maxsup$, $absmin$

Input

P Itemset prefix
λ Initial constructed RP-tree
C Apx-conditional patterns
R Captured ARI itemsets

Output

Approximate rare-item itemsets

1: **if** $P = \emptyset$ **then**
2: **for** rare item $i \in \lambda$ **do**
3: $C \leftarrow C \cup$ conditionals of i
4: convert C to Apx-conditional patterns ▷ No Propagation
5: **if** $Sup(X_p) \geq minsup$ **then**
6: print X_p
7: **end if**
8: ARI-Growth(X_p, λ, C, R)
9: **end for**
10: **else**
11: H = Header of C ▷ Recursive step. Test supersets
12: **for** $X \in H$ *where* $X.support > absmin$ **do**
13: $c \leftarrow$ conditionals of X
14: convert c to apx-conditional patterns
15: insert c into C ▷ Propagates
16: $X_p = X \cup$ P
17: update C apx-patterns as X_p ▷ Creates a copy of C
18: **if** $Sup(X_p) > absmin$ **then**
19: **if** $Sup_{apx}(C) < maxsup$ && $Sup_{apx}(C) \geq minsup$ && $\forall x \in X_p$, x.support in approximately supporting transactions is at least $\epsilon_c * Sup_{Apx}(C)$ **then**
20: print X_p
21: **end if**
22: ARI-Growth(X_p, λ, C)
23: **end if**
24: **end for**
25: **end if**

Upwards Propagation Example

For the purpose of example, let us suppose there exists an itemset $\{C, D\}$, the initial RP-tree given in Figure 3.3, and that $\{C, D\}$ was formed from having item C inserted to the apx-patterns of item D.

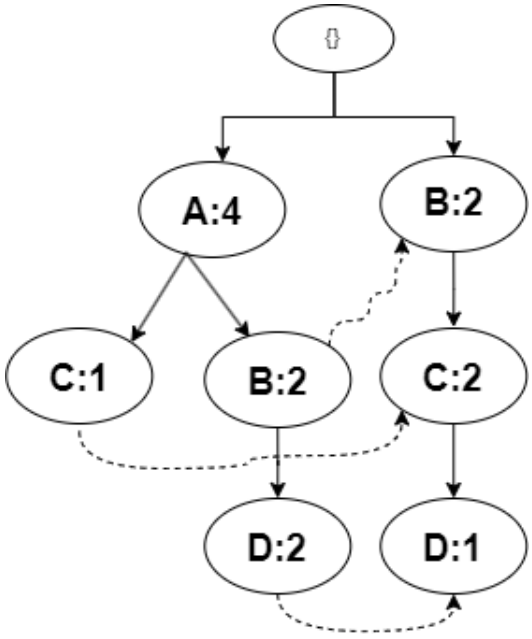

Figure 3.3: Example tree.

The apx-patterns of $\{C, D\}$ are as follows, item C: $\langle$ $\{A\}$, sup:1, $\epsilon_r = 50\%$, C:1 D:0 $\rangle$, $\langle$ $\{B\}$, sup:2, $\epsilon_r = 50\%$, C:2 D:0 $\rangle$, item D: $\langle$ $\{A, B\}$, sup:2, $\epsilon_r = 50\%$ C:0 D:2 $\rangle$, and $\langle$ $\{B\}$, sup:1, $\epsilon_r = 100\%$ C:1 D:1 $\rangle$. Item D succeeds item C. Suppose propagation continues upwards from component (6) of the apx-pattern for item D. Only one apx-pattern of item D contains C (i.e., C in component (5) is greater than 0), and thus, there is one pattern that can possibly reduce the support of item C's apx-pattern. Following component (6) of apx-pattern beginning from the tree node $D : 1$. Moving up to $C : 2$, C is reached. Thus, the apx-pattern associated with

node $C : 2$ for item C has the support of components (3) and (5) subtracted by the traversing apx-pattern (i.e., subtract support by 1). Next, component (6) of the traversing apx-pattern is replaced to now reference the tree node $C : 2$. This process is demonstrated in Figure 3.4.

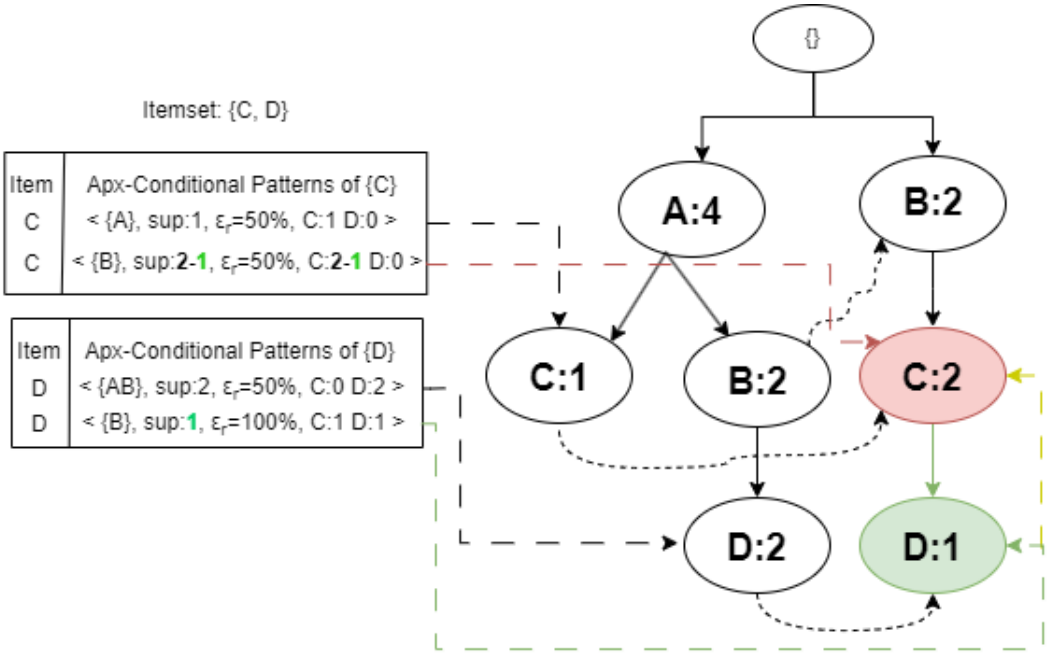

Figure 3.4: Upwards propagation of D for itemset $\{C, D\}$.

Let us further consider the insertion of item B into $\{C\}$ (i.e., itemset $\{B, C, D\}$). The newly inserted apx-patterns are, item B: $\langle$ $\{A\}$, sup:2, $\epsilon_r = \frac{1}{3}\%$, B:2 C:0 D:0 $\rangle$ and $\langle$ $\{\}$, sup:2, $\epsilon_r = \frac{1}{3}\%$, B:2 C:0 D:0 $\rangle$. The apx-patterns of itemset $\{C, D\}$ are as follows, item C: $\langle$ $\{A\}$, sup:1, $\epsilon_r = \frac{1}{3}\%$, B:0 C:1 D:0 $\rangle$, $\langle$ $\{\}$, sup:1, $\epsilon_r = \frac{2}{3}\%$, B:1 C:1 D:0 $\rangle$, item D: $\langle$ $\{A\}$, sup:2, $\epsilon_r = \frac{1}{3}\%$ B:2 C:0 D:2 $\rangle$, and $\langle$ $\{\}$, sup:1, $\epsilon_r = 100\%$ B:1 C:1 D:1 $\rangle$. There are three apx-patterns that contain B. For reference, this process is demonstrated in Figure 3.5 for each of the three apx-patterns. For the first apx-pattern (1) following component (6) of item C's apx-pattern beginning from

the tree node $C:2$. Moving up to node $B:2$, B is reached. The support of the connected apx-pattern of item B to node $B:2$ is subtracted by 1, and component (6) of the traversing apx-pattern is replaced with the last traversed tree node. For the next apx-pattern (2), follow component (6) of item D's apx-pattern beginning from the tree node $D:2$. Moving up to node $B:2$, B is reached. The support of the connected apx-pattern of item B to node $B:2$ is subtracted by 2, and component (6) of the traversing apx-pattern is replaced with the last traversed tree node. Finally, for the apx-pattern (3), follow component (6) of item D's apx-pattern beginning from the tree node $C:2$. Moving up to node $B:2$, B is reached. The support of the connected apx-pattern of item B to node $B:2$ is subtracted by 1, and component (6) of the traversing apx-pattern is replaced with the last traversed node.

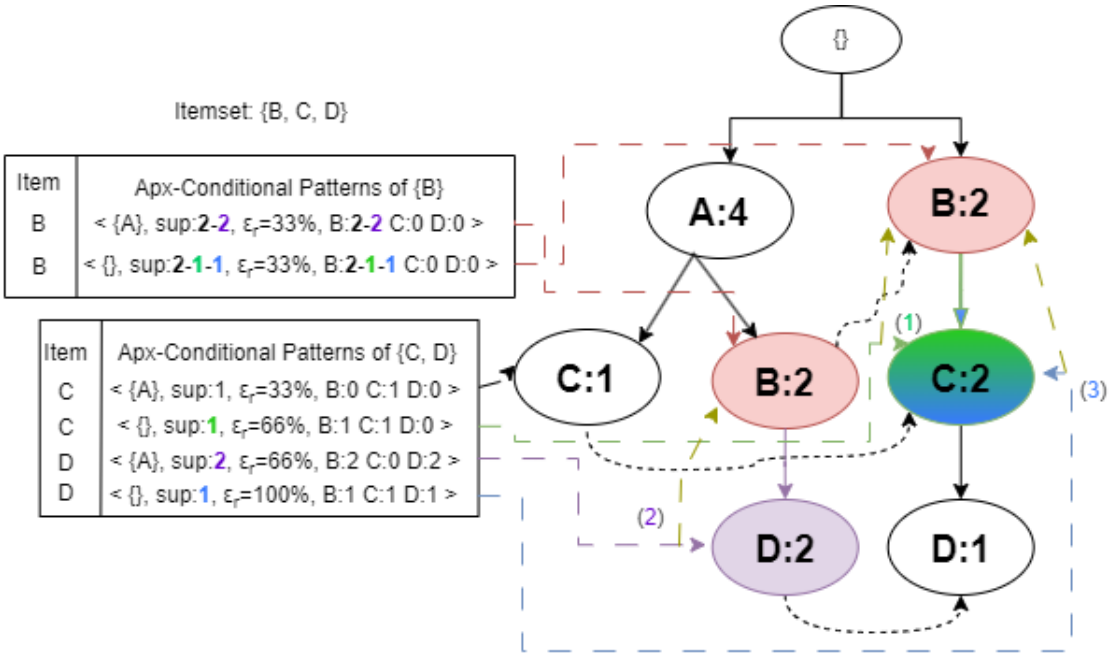

Figure 3.5: Upwards propagation of C & D for itemset $\{B, C, D\}$.

ARI-Growth Example

Let $minsup = absmin = \mathbf{1}$, $maxsup = \mathbf{6}$, $\epsilon_r = \mathbf{50\%}$, $\epsilon_c = \mathbf{50\%}$. For simplicity of the example, the satisfaction of *absmin* (i.e., $Sup(X) > absmin$) also satisfies *minsup*. The algorithm continues through Table 3.1 as follows. Size-1 itemsets are determined from a database scan as $\{A\} : 4$, $\{B\} : 4$, $\{C\} : 3$, $\{D\} : 2$, $\{E\} : 1$, of which, $\{E\}$ is ignored for failing to satisfy the *absmin* threshold ($1 \not> absmin$). The database is scanned again for the construction of an RP-tree. For each transaction, for each size-1 ARI in their respective order, the item is inserted into the tree. Figure 3.2 denotes the resulting RP-tree with its corresponding header table.

This algorithm discovers the complete set of interesting rare-item itemsets with the use of all rare size-1 and the initial RP-tree. For reference, component (6) of an apx-pattern is represented as an *edge* from an apx-pattern to a tree node.

Starting with $\{A\}$, conditional patterns of item A, $\{A\} : 4$ are generated. The apx-patterns are formed as follows, item A: $\langle$ {}, sup:4, $\epsilon_r = 100\%$, A:4 $\rangle$. $\{A\}$ is an ARI. Since there are no supersets to any apx-patterns, the algorithm moves to the next size-1 itemset.

Next, conditional patterns of item B, $\{AB\} : 2$, and $\{B\}$:2 are generated. The apx-patterns are formed as follows, item B: $\langle$ $\{A\}$, sup:2, $\epsilon_r = 100\%$, B:2 $\rangle$, and $\langle$ {}, sup:4, $\epsilon_r = 100\%$ B:2 $\rangle$. Figure 3.6 demonstrates the creation of these apx-patterns. $\{B\}$ is an ARI. The algorithm then generates a new header table by components (2), (3), and (4) of each apx-pattern by summing the support of each pattern that has $\epsilon_r = 100\%$ for each item in component (2). As a result, The header table $\{A\}$:2 is formed.

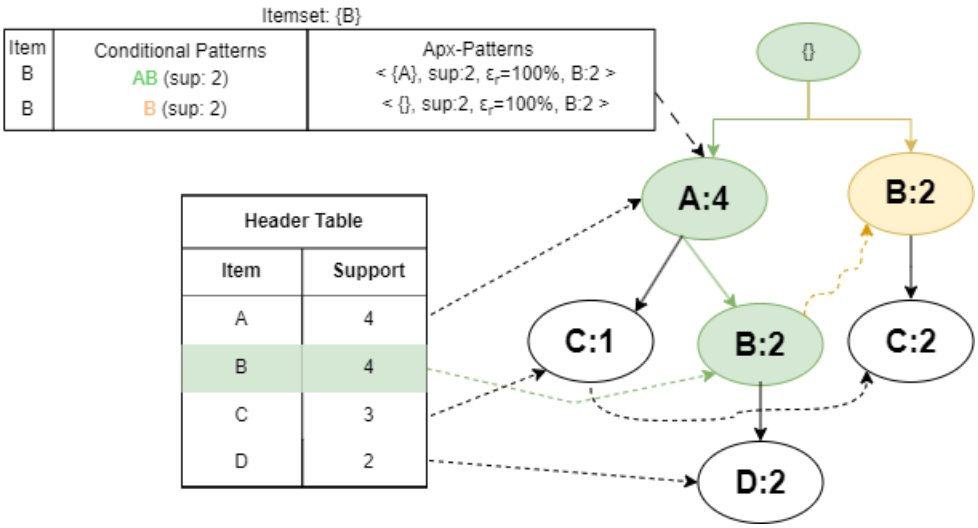

Figure 3.6: Apx-patterns of $\{B\}$

Going through the header table, $\{A\}$, the conditional patterns $\{A\} : 4$ are generated. The apx-patterns of item A are formed as follows, item A: $\langle$ {}, sup:4, $\epsilon_r = 100\%$, A:4 $\rangle$. Apx-patterns of item A are inserted to apx-patterns of item B to formulate apx-patterns of itemset $\{A, B\}$, with upwards propagation performed. The resulting apx-patterns of $\{A, B\}$ are formed as follows, item A: $\langle$ {}, sup:2, $\epsilon_r = 50\%$, A:2 B:0 $\rangle$. Item B: $\langle$ {} sup:2, $\epsilon_r = 50\%$, A:0 B:2 $\rangle$, and $\langle$ {} sup:2, $\epsilon_r = 100\%$, A:2 B:2 $\rangle$. $Sup_{apx}(\{A, B\} = 6\,,\, 6 \geq maxsup$, thus $\{A, B\}$ is not an ARI. Header table creation of item $\{B\}$ and generation of apx-patterns for itemset $\{A, B\}$ is demonstrated in Figure 3.7. The header table {} is formed. As there are no items in the header table, the recursive step ends. The algorithm backtracks to itemset B, and since there are no remaining items in the header table of $\{A\}$, the algorithm moves to the next size-1 itemset.

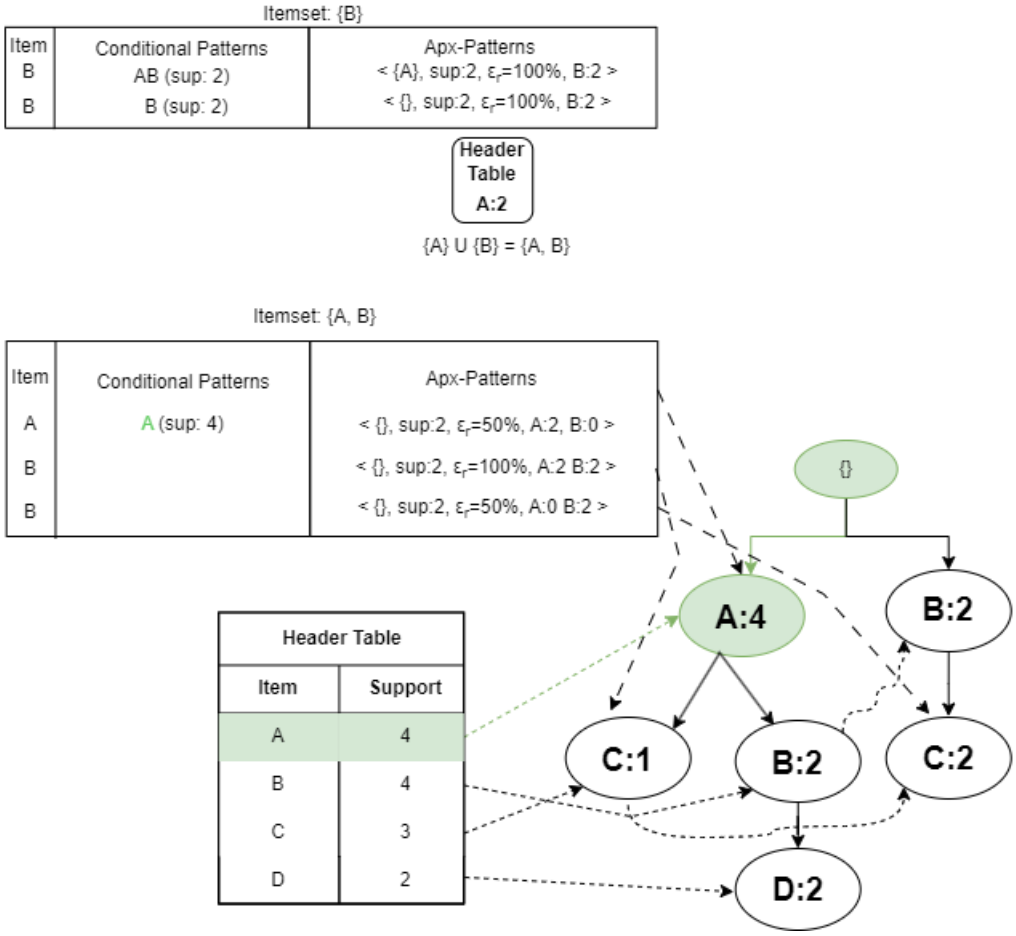

Figure 3.7: Apx-patterns of $\{A, B\}$

Next, conditional patterns of item $\{C\}$, $\{AC\} : 1$, and $\{BC\} : 2$ are generated. The apx-patterns of item C are formed as follows, item C: $\langle$ $\{A\}$, sup:1, $\epsilon_r = 100\%$, C:1 $\rangle$, and $\langle$ $\{B\}$, sup:2, $\epsilon_r = 100\%$, C:2 $\rangle$. $\{C\}$ is an ARI. The header table $\{B\} : 2$ is formed. Since $\{A\} : 1 \leq absmin$, $\{A\}$ is excluded from the header table.

Going through the header table, B conditional patterns $\{A, B\} : 2$, $\{B\} : 2$ are generated. The apx-patterns of item B are formed as follows, item B: $\langle$ $\{A\}$, sup:2, $\epsilon_r = 100\%$, B:2 $\rangle$, and $\langle$ $\{\}$, sup:2, $\epsilon_r = 100\%$, B:2 $\rangle$. Apx-patterns of item B are

inserted to apx-patterns of item C to formulate apx-patterns of itemset $\{B, C\}$, with upwards propagation performed. The resulting apx-patterns of $\{B, C\}$ are formed as follows, item B: $\langle$ $\{A\}$, sup:2, $\epsilon_r = 50\%$, B:2, C:0 $\rangle$, and item C: $\langle$ $\{A\}$, sup:1, $\epsilon_r = 50\%$, B:0 C:1 $\rangle$, and $\langle$ $\{\}$, sup:2, $\epsilon_r = 100\%$, B:2 C:2 $\rangle$. $Sup_{apx}(\{B, C\} = 5$, $5 < maxsup$. Similarly, $Sup(\{B, C\} = 2$, $2 > absmin$. $\{B, C\}$ is checked to satisfy ϵ_C, of which, C appears in 3 transactions ($\frac{3}{5} > \epsilon_c$), and B appears in 4 transactions ($\frac{4}{5} > \epsilon_c$). Thus, $\{B, C\}$ is an ARI. The header table $\{\}$ is formed. Since $\{A\} : 1 \leq absmin$, $\{A\}$ is excluded from the header table. As there are no more items in the header table, the recursive step of $\{B, C\}$ ends. The algorithm backtracks to itemset $\{C\}$. As there are no remaining items in the header table of $\{B\} : 2$, the algorithm moves to the next size-1 itemset.

Finally, conditional patterns of item D, $\{A, B, D\} : 2$ are generated. The apx-patterns are formed as follows, item D: $\langle$ $\{A, B\}$, sup:2, $\epsilon_r = 100\%$, D:2 $\rangle$. $\{D\}$ is an ARI. The header table $\{A\} : 2$, $\{B\} : 2$ is formed. Going through the header table starting from $\{A\} : 2$, A conditional patterns $\{\}$ are generated. The apx-patterns of item A are formed as follows, item A: $\langle$ $\{\}$, sup:4, $\epsilon_r = 100\%$ A:4 $\rangle$. The apx-patterns of item A are inserted into apx-patterns of item D to formulate apx-patterns of itemset $\{A, D\}$, with upwards propagation performed in additional to the removal of any item in component (1) larger than A. The resulting apx-patterns of $\{A, D\}$ are formed as follows, item A: $\langle$ $\{\}$, sup:2, $\epsilon_r = 50\%$, A:2 D:0 $\rangle$. Item D: $\langle$ $\{\}$ sup:2, $\epsilon_r = 100\%$, A:2 D:2 $\rangle$. There are 4 approximately supporting transactions, and 2 are fully supporting. Thus, itemset $\{A, D\}$ satisfies both the $maxsup$ and $absmin$ ($Sup_{apx}(\{A, D\}) < maxsup$, and $Sup(\{A, D\}) > absmin$). $\{A, D\}$ is checked to

satisfy ϵ_c. A occurs a total of 4 times ($100\% \geq \epsilon_c$), D occurs 2 times ($50\% = \epsilon_c$). Thus, $\{A, D\}$ is an ARI. The formation of apx-patterns for itemset $\{A, D\}$ is demonstrated in Figure 3.8. The header table $\{\}$ is formed from apx-patterns of $\{A, D\}$. As there are no more items in the header table, the recursive step of $\{A, D\}$ ends. The algorithm backtracks to itemset $\{D\}$. As the header table contains $\{A\} : 2$, $\{B\} : 2$, the algorithm moves to $\{B\}$.

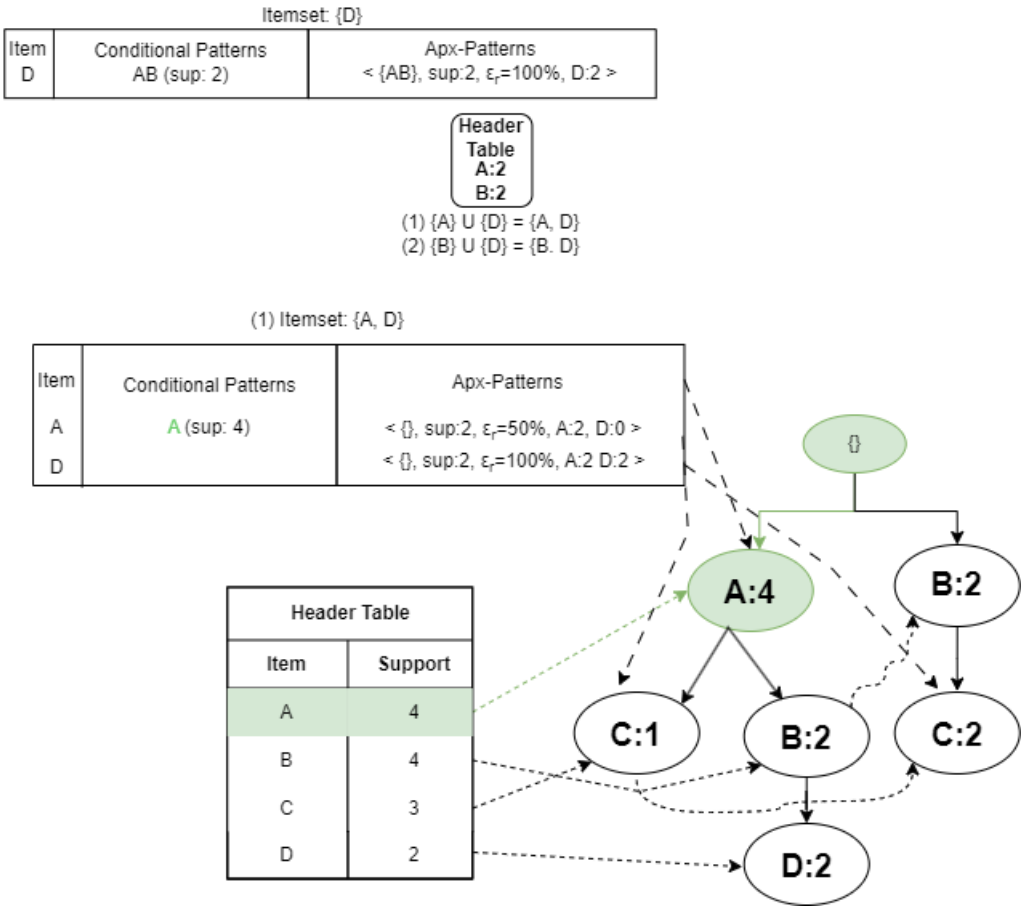

Figure 3.8: Apx-patterns of $\{A, D\}$

From $\{B\}$, conditional patterns $\{A, B\} : 2$, and $\{B\} : 2$ are generated. The apx-patterns of item B are formed as follows, item B: $\langle$ $\{A\}$, sup:2, $\epsilon_r = 100\%$, B:2 $\rangle$, and $\langle$ $\{\}$, sup:2, $\epsilon_r = 100\%$, B:2 $\rangle$. The apx-patterns of item B are inserted into apx-patterns of item D to formulate apx-patterns of itemset $\{B, D\}$, with upwards propagation performed. The resulting apx-patterns of $\{B, D\}$ are formed as follows, item B: $<$ $\{\}$, sup:2, $\epsilon_r = 50\%$, B:2 D:0 $\rangle$, item D: $<$ $\{A\}$ sup:2, $\epsilon_r = 100\%$, B:2 D:2 $\rangle$. $Sup_{apx}(\{B, D\}) = 4$, $4 < maxsup$. Similarly, $Sup(\{B, D\}) = 2, 2 > absmin$. $\{B, D\}$ is checked to satisfy ϵ_c. Item B appears in 4 transactions ($\frac{4}{4} > \epsilon_c$), and item D appears in 2 ($\frac{2}{4} = \epsilon_c$). Thus, $\{B, D\}$ is an ARI. The header table $\{A\} : 2$ is formed.

Going through the header table, A conditional patterns $\{A\} : 4$ are generated. The apx-patterns of item A are formed as follows, item A: $\langle$ $\{\}$, sup:4, $\epsilon_r = 100\%$, A:4 $\rangle$. Apx-patterns of item A are inserted to apx-patterns of $\{B, D\}$ to formulate apx-patterns of itemset $\{A, B, D\}$, with upwards propagation performed. The resulting apx-patterns of itemset $\{A, B, D\}$ are formed as follows, item B: $\langle$ $\{\}$, sup:2, $\epsilon_r = 33\%$, A:0 B:2 D:0 $\rangle$, item A: $\langle$ $\{\}$, sup:2, $\epsilon_r = 33\%$, A:2 B:0 D:0 $\rangle$. Item D: $\langle$ $\{\}$ sup:2, $\epsilon_r = 100\%$, A:2 B:2 D:2 $\rangle$. Redundant patterns (i.e., $\epsilon_r < 0.5 \& component(3) = \{\}$) are ignored in this process in order to yield Item D: $\langle$ $\{\}$ sup:2, $\epsilon_r = 100\%$, A:2 B:2 D:2 $\rangle$. There are 2 fully and approximately supporting transactions (i.e., $Sup_{apx}(\{A, B, D\}) < maxsup$, and $Sup(\{A, B, D\} > absmin)$. $\{A, B, D\}$ is checked to satisfy ϵ_C, of which, all items appear in all transactions ($100\% > \epsilon_c$). Thus, $\{A, B, D\}$ is an ARI. The header table $\{\}$ is formed. As there are no items in the header table, the recursive step ends. The algorithm backtracks to itemset $\{B, D\}$. As there are no remaining items in the header of $\{A\} : 2$, the algorithm backtracks to

itemset D. Similarly, there are no remaining items in the header of $\{A\} : 2$, $\{B\} : 2$, and thus the algorithm moves to the next size-1 itemset. Since $\{D\}$ was the final size-1 itemset, the algorithm terminates.

Minsup Pruning

Global pruning is employed [LPS$^+$06] to prune itemsets by the *minsup* threshold. By ϵ_c, in order for an itemset to be interesting, the overall support of each item of an itemset must exceed $minsup * \epsilon_c$. Let us suppose there exists an itemset $\{A, B\}$. In order for $\{A, B\}$ to be interesting, the itemsets approximate support must exceed *minsup*; i.e., $Sup_{apx}(\{A, B\}) \geq minsup$. By ϵ_c, this implies $\{A\}$ and $\{B\}$ occur at least $\epsilon_c * minsup$ times. If this fails, no supersets of $\{A, B\}$ will exceed the required ϵ_c threshold. To allow for pruning, the recursive step in line 22 of Algorithm 1 is prevented if global pruning fails.

Caching

Caching is a process of temporarily storing information for later use, to prevent the need of re-calculating or re-creating the needed data. In ARI-growth, conditional patterns are generated for all items of an itemset. Constant generation of conditional patterns results in excess computation time. As a result, conditional patterns of size-1 items are cached for later use after they are generated once for the first time.

Algorithmic Adjustments

The proposed algorithm is introduced with respect to discovering rare-item itemsets. This algorithm can however be adjusted to accommodate other definitions of rare itemsets such as sporadic rules defined in Section 2.2. An implementation may vary depending on how one may wish to accommodate noise. To discover sporadic

rules whilst maintaining a *maxsup* subset definition where all subsets are rare by the *maxsup*, the recursive step in line 22 in Algorithm 1 can be adjusted to ensure the recursive step ends if the approximate support is larger than *maxsup*. After algorithm execution, an additional pruning step is performed to ensure all subsets of an ARI are also an ARI.

3.4 Summary

In this chapter, the problem and solution of mining rare itemsets in the presence of noise is introduced. Many research areas consider the discovery of uncommonly occurring patterns [BN19]. However, in the presence of noise, interesting uncommon patterns may be lost. For instance, in a real-world retail dataset, noise heavily influences the number of sporadic rules, of which, are association rules with rare itemsets that maintain a high confidence. To address this problem, the previously proposed approximate model for the discovery of core itemsets is modified for the discovery of rare itemsets. In order to discover an approximately rare itemset, two algorithms are introduced. A Naive algorithm is the simplest approach to introduce a problem solution. However, the simplest approaches are often not useful in the real world, and in this case, all possible combinations of itemsets will be generated. As such, in this thesis, I introduce a more practical pattern growth approach to discover rare itemsets more efficiently. The proposed approach generates a tree structure similar to that of an RP-tree [TKD11] and then generates all rare itemsets from this tree. Similar to the Apx-pattern growth [BL21], apx-patterns are created to discover interesting itemsets. The proposed ARI-growth algorithm improves upon the Apx-pattern growth by

avoiding the generation of size-2 itemsets and by creating one tree.

Chapter 4

Evaluation

In this chapter, the proposed ARI-growth algorithm's effectiveness is evaluated. Since the anti-monotone property does not hold in the case of approximately rare itemsets, approximate mining is inherently a more computationally expensive task. That is, pruning of itemsets must be more lenient to that of *exact* mining. To measure how much more computationally expensive approximate mining is, the effectiveness of ARI-growth is evaluated in terms of run-time and memory consumption in comparison to that of *exact* mining techniques. Furthermore, in order to evaluate if the algorithm is an applicable approximation model in the presence of noise, correctness in obtaining the true set of patterns is evaluated under precision and recall criteria.

4.1 Evaluation Setup

Using well-known rare pattern mining algorithms presents an idea of how much better or worse the presence of noise poses on computational constraints in order to

capture uncommonly occurring patterns. In this thesis, rare pattern mining algorithms Apriori-inverse (Section 2.2.3) and RP-growth (Section 2.2.4) are used as a baseline. During testing, the Naive algorithm showed extensively slow execution times and thus has been dropped for consideration. As a result, I deem the Naive approach unsuitable in real-world datasets. Characteristics of each algorithm are summarized in Table 4.1.

Table 4.1: Algorithm characteristics.

Algorithm	Scalable	Tree-based	Rare Patterns	Approximation
Naive			✓	✓
Apriori-Inverse	✓		✓	
RP-Growth	✓	✓	✓	
ARI-Growth	✓	✓	✓	✓

To ensure results of all algorithms can be made equal, Apriori-inverse is modified to discover rare-item itemsets as described in Section 2.2.2. A fixed-threshold [KR05] is used, where *maxsup* is increased during the generation of candidates and set normally for rule generation in order to discover rare itemsets that are not limited to perfectly sporadic rules described in section 2.2.1. $Maxsup$ is thus temporarily set to infinity. When all itemsets with support larger than *absmin* are discovered, itemsets with support that does not meet the original *maxsup* criteria are pruned (i.e., $Support(X) < maxsup$). Furthermore, an additional database scan is performed. The additional database scan stores the set of all transactions t that contain at least one itemset in L1. During each database scan, Apriori-inverse only scans transactions in t, as opposed to scanning the whole dataset. Whilst Apriori-inverse and FP-growth apply the optimization of scanning transactions in t, the proposed ARI-growth can-

not apply this optimization when ϵ_r, $\epsilon_c < 1$. This is because a transaction that only contains frequent itemsets can approximately support an itemset that contains a rare item. Since ARI-growth applies approximation only when $\epsilon_r \,\&\, \epsilon_c < 1$, scanning of t transactions is not applied to the ARI-growth. Single-path optimization is applied to RP-growth [TKD11]. For future reference, Apriori-inverse modifications are denoted as the algorithm Apriori-inverse*.

All experiments are run on an AMD Ryzen 7 2700X with a base processor speed of 3.70 GHz and 16GB of RAM. Each algorithm is implemented in Java. 1024MB of heap memory is provided to Java Virtual Machine. Three real databases are used as a benchmark. All datasets are commonly used in pattern mining and are chosen to match those previously used in noisy dataset experiments [BL21]. The real world *retail* [1] dataset is obtained from the frequent itemset mining repository (http://fimi.ua.ac.be). The remaining *BMSWebView1* and *foodMart* datasets are obtained from the SPMF data mining library [FLG+16]. Characteristics of experimented datasets are provided in Table 4.2, with characteristics of transaction lengths in Table 4.3. As shown in Tables 4.2 and 4.3, *retail* contains longer transactions on average in comparison to *BMSWebView1* and *foodmart*, and more transactions overall, and *BMSWebView1* has a lesser number of unique items.

[1] Special thanks to Tom Brijs for contributions in the production of the retail dataset [BSVW99]. As a result of the work that creates public datasets, algorithms are benchmarked in a manner that can be replicated.

Table 4.2: Dataset characteristics.

Database	Transactions	Unique Items	Density (%) [2]	Size (KB)
Retail	88,162	16,470	0.06	4,070
BMSWebView1	59,602	497	0.51	935
FoodMart	4,141	1,559	0.28	81

Table 4.3: Transaction length characteristics.

Database	Mean	Median 50th (25th - 75th)	Standard Deviation	max
Retail	10.3	8 (4 - 14)	8.16	76
BMSWebView1	2.5	1 (1 - 3)	4.85	267
FoodMart	4.4	4 (3 - 6)	2.11	14

4.1.1 Evaluation of Run-time

In this section, the algorithm run-time is evaluated to address the following two questions in Section 1.2.1 *Can itemsets that are approximated be discovered efficiently to be applicable in the real world? How does efficiency compare to that of traditional mining techniques?*. During experiments, Apriori-inverse* yields extensively slow computation time on datasets with long transaction lengths, such as retail. Executions of Apriori-inverse* have been halted for these datasets, with the algorithm deemed unfit for solving the task. Experimental results of processing time are provided in Figures 4.1, 4.2, and 4.3. ARI-growth is executed with varying ϵ_r and ϵ_c thresholds, to observe if threshold adjustments have an effect on the run-time of the algorithm. ϵ_r and ϵ_c of 0.5, 0.75, and 1.0 are tested where $\epsilon_r = \epsilon_c$. Run-time is measured by subtracting the time after the algorithm from the time before the algorithm

[2]The mean number of items per transaction divided by the total number of unique items in the database. Density is provided by the SPMF data mining library [FLG+16].

execution. This measure is subject to inaccuracies imposed by the Windows operating system, in addition to the possible execution of garbage collection from Java Virtual Machine.

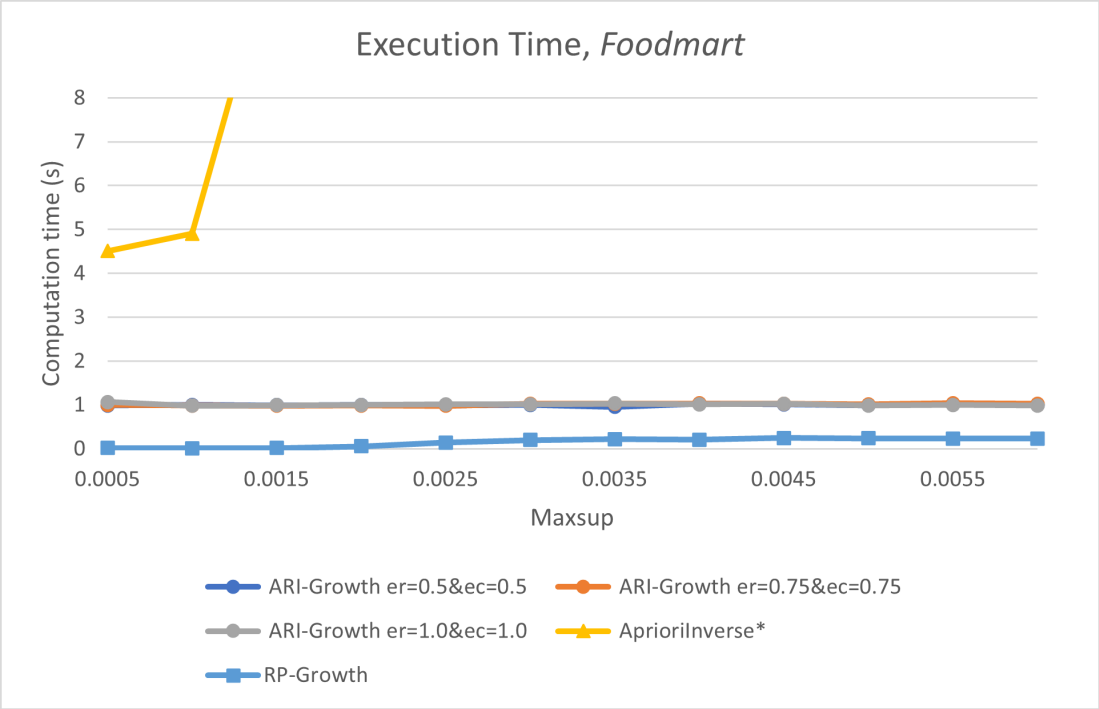

Figure 4.1: Foodmart execution time. $Minsup = absmin = 0.0005$.

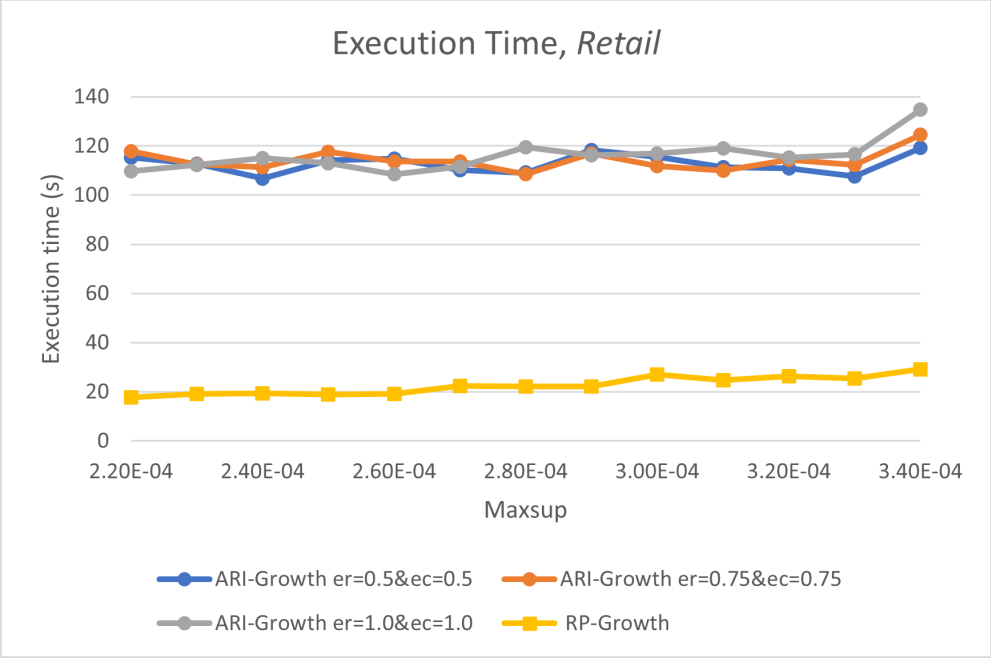

Figure 4.2: Retail execution time. $Minsup = absmin = 0.00007$.

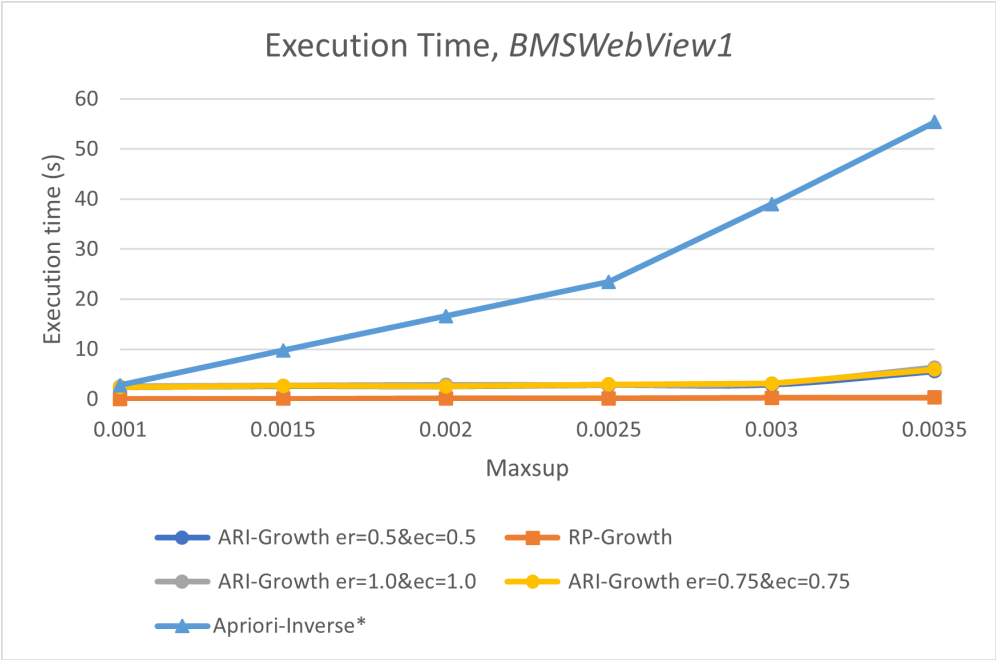

Figure 4.3: BMSWebView1 execution time. $Minsup = absmin = 0.00075$.

In the *foodmart* dataset, as a result of the high number of unique items proportional to the length of the dataset, in addition to a mean transaction length of 4, resulting patterns are observed to be very short (less or equal to a length of 2). Performance is largely compared in terms of the initial stages of an algorithm. As shown in Figure 4.3, the proposed model is an *order of magnitude* faster than Apriori-inverse*, whilst at the same time, far slower than that of RP-growth. In Figure 4.1, Apriori-inverse* is trimmed as it continues with a long execution time at a $maxsup$ of 0.002. Similar results are shown in the *retail* dataset in Figure 4.2.

In the *BMSWebView1* dataset, Apriori-inverse* has an efficient computation time. *BMSWebView1* contains a small number of unique items proportionally to the total number of transactions, and thus often results in lesser size-1 itemsets with longer rare itemsets being discovered. An Apriori approach performs well in this application because the number of candidates generated depends on the number of interesting patterns of the previously discovered length. In the case of *foodmart*, a large number of size-2 candidate itemsets will be generated. Despite the efficiency of Apriori-inverse* within this application, the proposed algorithm outperforms Apriori-inverse* by a significant margin as shown in Figure 4.3. Overall, the proposed algorithm is effective in real-world datasets, executing to completion and in all cases, outperforming Apriori-inverse*. Results are observed to be similar when approximation thresholds are varied.

4.2 Evaluation of Memory Consumption

In this section, memory consumption is evaluated to address the following two questions in Section 1.2.1 *Can itemsets that are approximated be discovered efficiently? How does efficiency compare to that of traditional mining techniques?*. Experimental results are provided in Figures 4.4 and 4.4. Similar to run-time comparisons, ARI-growth is executed with varying ϵ_r and ϵ_c thresholds, to observe if threshold adjustments have an effect on the memory consumption of the algorithm. ϵ_r and ϵ_c of 0.5, 0.75, and 1.0 are tested where $\epsilon_r = \epsilon_c$. Memory consumption is measured by approximating the total space consumed by objects from Java Virtual Machine. This measure is subject to inaccuracies imposed by Java Virtual Machine's garbage collection. In *retail* test runs, Apriori-inverse* runs out of heap memory. As a result of garbage collection inconsistencies, *retail* is excluded from graphical memory comparisons. Since ARI-growth executes to completion, ARI-growth still exceeds Apriori-inverse*.

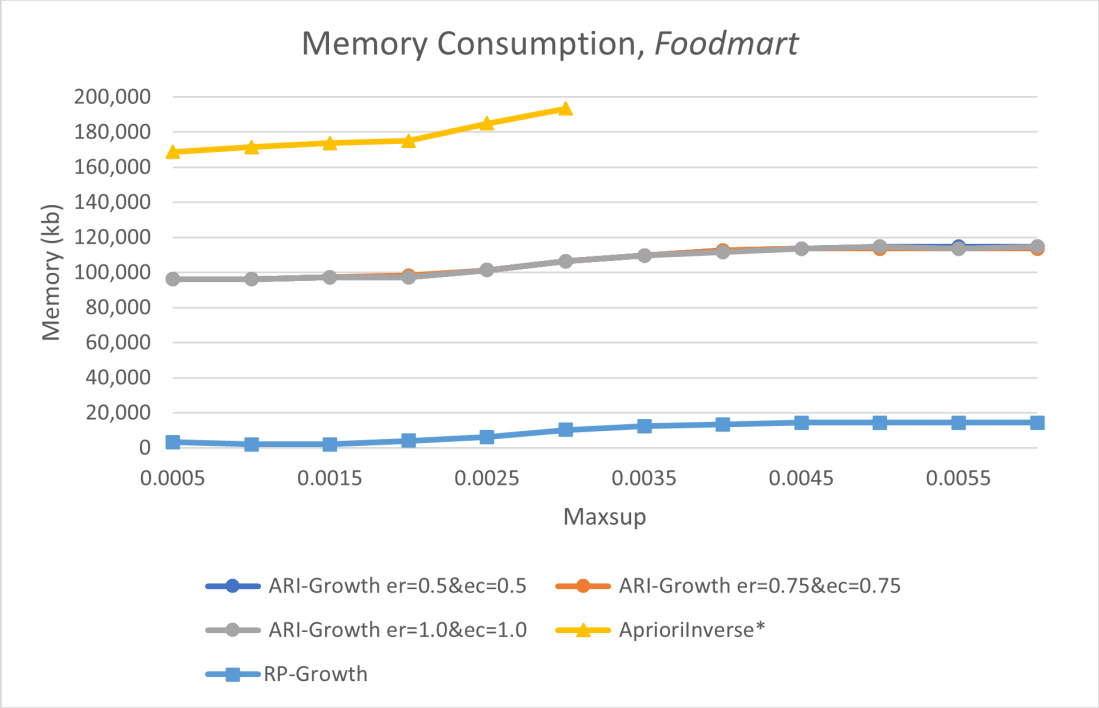

Figure 4.4: Foodmart memory consumption. $Minsup = absmin = 0.0005$.

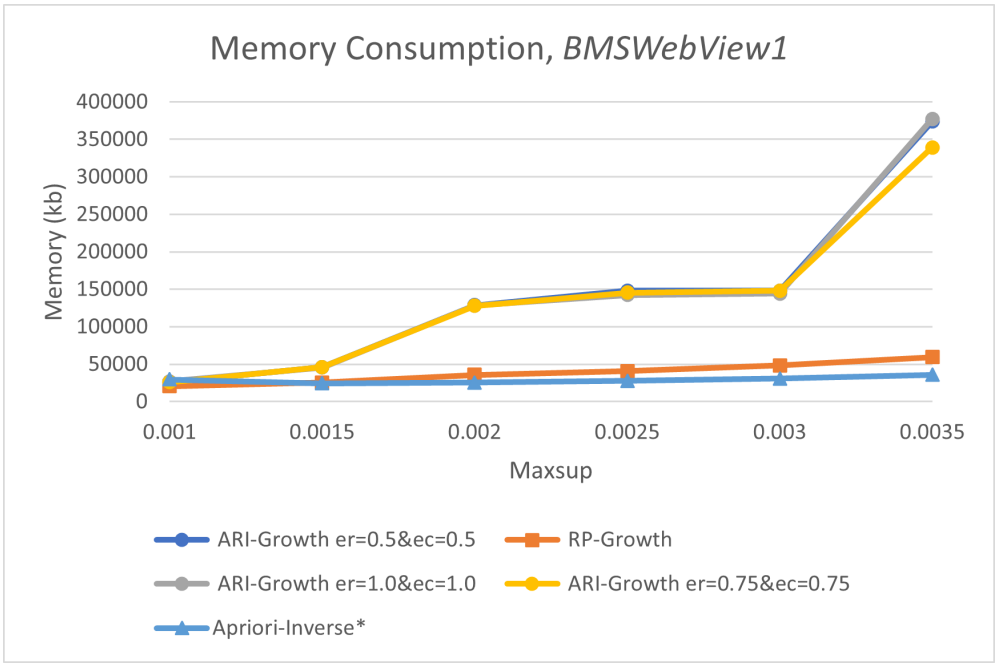

Figure 4.5: BMSWebView1 memory consumption. $Minsup = absmin = 0.00075$.

In the case of *foodmart* shown in Figure 4.4, the proposed model outperformed Apriori-inverse*, whilst also outperformed by RP-growth. Additionally, from *BMSWebView1* in Figure 4.4, the proposed model performs the worst, having a significantly higher memory consumption. Considering the amount of memory consumed, both datasets yield larger than 100 megabytes. In the case of different approximation thresholds, results follow similar trends. Whilst comparisons of execution times for the proposed model largely remain the same, the number of captured itemsets differs, and thus, may influence changes to memory consumption. In the case of *foodmart*, the vast majority of itemsets are not long enough for the approximation model to take effect. For instance, with $maxsup = 1.0$ and $absmin = 0.0005$, 1635 itemsets are captured; 1557 of which, are of size-1. Results are similar under varied approximation thresholds. Overall, the proposed model is constrained by memory, as consumption is very high for *BMSWebView1* when multiple recursive steps for each size-1 itemset take place to discover longer itemsets.

4.3 Evaluation of Recovery Quality

In this section, the quality of pattern recovery is evaluated to address the question proposed in Section 1.2.1 *Can approximation be applied to recover lost, yet interesting, rare itemsets? Can the same approximation of itemsets be applied to recover lost, yet interesting, association rules?*. To evaluate recovery, ARI-growth is compared under a variety of ϵ_r and ϵ_c thresholds with varied levels of noise. Quality is tested with

precision and recall metrics utilized by Cheng *et al.,* [CYH08]:

$$Precision = \frac{|true \cap aprx|}{|aprx|} \tag{4.1}$$

$$Recall = \frac{|true \cap aprx|}{|true|} \tag{4.2}$$

where *true* denotes the set of true core patterns that exist from traditional rare pattern techniques under the assumption of full noise recovery, and *aprx* denotes the set of patterns determined by the proposed algorithm. To apply $x\%$ noise, each item is assigned $x\%$ chance of no longer existing (i.e., information is lost) or, for each item in the dataset, there is an $x\%$ chance of a new item appearing in the transaction (i.e., information is gained). The odds of applying noise is a uniform random distribution. Dataset *BMSWebView1* is tested under thresholds $minsup = 0.00175$, $absmin = 0.00125$, $maxsup = 0.045$, and *retail* under thresholds $minsup = 0.00175$, $absmin = 0.0003$, $maxsup = 0.05$. Tests utilize varying row and error tolerance thresholds, precision results are shown in Tables 4.4 & 4.6, and recall in Tables 4.5 & 4.7.

In the tested environments improvements to precision yield at most a 0.2% improvement. In *BMSWebview1* precision improvements are observed when 15% or more noise is attributed to the dataset, and 5-15% in *retail*. In most cases, both precision and recall show similar or worse results than when no approximation is applied. Despite the overall lack of improvement when not applying approximation, pattern recovery is observed. Higher rates of allowed error tolerance results in greater recovery, however, typically has low overall precision and recall in contrast to lower rates of allowed error tolerance. For instance, in *retail* with $\epsilon_r, \epsilon_c = 0.25$, up to 6%;

Table 4.4: Pattern precision *BMSWebView1* (%).

Noise (%)	ϵ_r, ϵ_c 1.0	0.8	0.75	$\frac{2}{3}$	0.5	$\frac{1}{3}$	0.25
5	**97.9**	**97.9**	**97.5**	**96.8**	**82.2**	67.5	64.2
10	94.0	94.0	94.0	94.0	**82.2**	70.8	68.1
15	90.4	90.5	90.5	90.5	82.0	77.8	76.6
20	86.6	86.7	86.8	86.8	77.5	**80.2**	**79.3**
25	79.6	79.8	79.8	79.8	69.1	76.3	77.5

Table 4.5: Pattern recall *BMSWebView1* (%).

Noise (%)	ϵ_r, ϵ_c 1.0	0.8	0.75	$\frac{2}{3}$	0.5	$\frac{1}{3}$	0.25
5	**85.0**	**85.0**	**83.6**	**84.0**	30.7	45.7	50.4
10	74.0	74.0	73.7	74.2	**30.8**	46.1	51.1
15	62.6	62.6	62.5	62.6	29.8	**46.3**	**51.6**
20	55.7	55.7	55.8	55.8	29.1	45.9	50.7
25	50.7	50.7	50.7	50.7	28.3	43.1	47.9

or 15, 23, 37, 37, and 39 rare patterns, that would otherwise be lost are recovered under the presence of 5%, 10%, 15%, 20% or 25% applied noise respectively.

While recovery of rare itemsets is important, a rare itemset by itself often requires a measure independent of support. For example, in the case of sporadic rules [KR05], an association rule associated with a rare itemset must maintain high confidence (i.e., often occurring together). Quality is further evaluated to that of resulting association rules. With $minconf = 85\%$, Precision through the use of approximate support is shown in Tables 4.8 and 4.10, and similarly recall in Tables 4.9 and 4.11.

By the aformentioned tables, under the tested conditions, precision sees no improved quality with any amount of applied approximation in *BMSWebView1* whilst up to a 1% in *retail*. In the case of recall, up to a 1% overall improvement is observed.

Table 4.6: Pattern precision *Retail* (%).

Noise (%)	ϵ_r, ϵ_c 1.0	0.8	0.75	$\frac{2}{3}$	0.5	$\frac{1}{3}$	0.25
5	**99.0**	**99.0**	**99.2**	**99.1**	62.3	32.5	27.1
10	98.9	98.9	99.1	**99.1**	65.8	36.0	30.5
15	98.8	98.8	98.9	98.9	71.9	39.7	33.3
20	98.8	98.8	98.8	98.8	75.8	44.1	37.1
25	98.7	98.7	98.7	98.7	**83.1**	**47.4**	**39.8**

Table 4.7: Pattern recall *Retail* (%).

Noise (%)	ϵ_r, ϵ_c 1.0	0.8	0.75	$\frac{2}{3}$	0.5	$\frac{1}{3}$	0.25
5	**87.3**	**87.1**	**85.1**	**85.1**	**36.3**	**36.7**	**36.7**
10	76.3	76.2	75.0	75.0	34.4	34.9	34.9
15	64.8	64.8	64.0	64.0	31.7	32.1	32.1
20	54.6	54.6	54.4	54.4	29.4	30.0	30.0
25	46.3	46.3	46.1	46.1	26.6	27.2	27.2

With a recall of $\epsilon_r, \epsilon_c = 0.25$, nearly $\frac{4}{5}$ or all of the association rules falsely represent true data in *BMSWebView1* or *retail* respectively. Despite this rate of errors, recovery is observed. For instance, with $\epsilon_r, \epsilon_c = 0.25$ up to 16%; or 112, 210, 286, 278, and 254 interesting association rules, that would otherwise be lost are recovered under the presence of 5%, 10%, 15%, 20% or 25% applied noise respectively with *BMSWebView1*. However, in the case of *retail*, less or equal to than 1% precision and recall is found under all tested thresholds $\epsilon_r, \epsilon_c \leq 0.5$. Or, similarly, in *BMSWebView1* less than 31% precision or 21% recall. As a result of the significant rates of incorrect classification for the true set of association rules, determining which association rules represent the true set of data requires additional effort.

Poor quality measures can be attributed to errors in the capability of a pattern's

Table 4.8: Association rule precision *BMSWebView1* (%). **Minconf = 85%**.

	ϵ_r, ϵ_c						
Noise (%)	1.0	0.8	0.75	$\frac{2}{3}$	0.5	$\frac{1}{3}$	0.25
5	**100.0**	**100.0**	89.5	69.3	19.9	15.3	15.0
10	94.5	94.5	90.4	77.6	27.2	18.9	18.1
15	94.2	94.2	88.5	84.1	**38.6**	24.4	24.0
20	94.5	94.5	86.9	86.9	32.4	28.5	27.3
25	93.4	93.4	**93.4**	**93.4**	28.9	**31.4**	**30.9**

Table 4.9: Association rule recall *BMSWebView1* (%). **Minconf = 85%**.

	ϵ_r, ϵ_c						
Noise (%)	1.0	0.8	0.75	$\frac{2}{3}$	0.5	$\frac{1}{3}$	0.25
5	**65.4**	**65.4**	**57.3**	**59.6**	3.5	14.8	20.0
10	45.9	45.9	44.0	46.9	**3.6**	15.1	20.2
15	31.6	31.6	31.0	31.5	2.7	**15.3**	**20.6**
20	22.7	22.7	23.1	23.1	1.7	14.6	18.8
25	18.4	18.4	18.4	18.4	1.0	12.4	16.5

approximate support to accurately predict the true support of a pattern. For example, let us consider an itemset X = $\{A, B, C\}$, Y = $\{B, C\}$, and Z = $\{A, C\}$. Let us further suppose $\epsilon_r = \frac{1}{3}$. In this case, both Z and Y contribute to the approximate support of X. For instance, if the support of Y is 4, the support of Z is 3, and the support of X is 1, then the approximate support of X is at least 6. ϵ_c is a measure used to trim patterns resulting from noise by ensuring each item of the itemset is present at least ϵ_c times the approximately supported transactions. Because approximate support is lenient to the original definition of support, it is always the case where $Sup_{apx}(X) \geq Sup(X)$. If the approximate support is not representative of noise, a significant *over estimate* to a pattern's true support occurs. In Table 4.8, correctness becomes more stable as noise increases. All thresholds $\epsilon_c, \epsilon_r < 1$ have a trend of gradually increased precision as

Table 4.10: Association rule precision *retail* (%). **Minconf = 85%**.

	ϵ_r, ϵ_c						
Noise (%)	1.0	0.8	0.75	$\frac{2}{3}$	0.5	$\frac{1}{3}$	0.25
5	**97.6**	**97.5**	**98.2**	97.5	0.1	0.1	0.1
10	96.8	96.7	97.8	**97.8**	0.1	0.2	0.2
15	96.7	96.7	97.0	97.0	**0.5**	0.4	0.3
20	96.7	96.7	96.6	96.6	0.0	0.6	0.4
25	96.2	96.2	96.1	96.1	0.0	**0.8**	**0.6**

Table 4.11: Association rule recall *retail* (%). **Minconf = 85%**.

	ϵ_r, ϵ_c						
Noise (%)	1.0	0.8	0.75	$\frac{2}{3}$	0.5	$\frac{1}{3}$	0.25
5	**77.6**	**74.8**	**60.5**	**60.5**	**0.2**	0.8	0.8
10	57.7	56.7	49.0	49.0	**0.2**	**1.0**	**1.0**
15	39.4	39.4	35.0	35.0	**0.2**	**1.0**	**1.0**
20	27.1	27.1	26.0	26.0	0.0	**1.0**	**1.0**
25	20.5	20.5	19.7	19.7	0.0	**1.0**	**1.0**

more noise is applied to the dataset. In the confidence measure of an association rule, the larger the estimate of the combined support for a rule, the larger the confidence. This is shown in Table 4.9 where the opposite effect of decreased recall occurs with increased noise. A greater number of *false positives* occur with increased noise.

To further show the incorrectness of approximated support for rule recovery, the *exact* support of an approximated pattern is used as opposed to approximate support in Table 4.12 to measure an association rules confidence. From the below table, it is observed that the approximate support fails to correctly capture a pattern's true support for accurate measurement of association rules. Through the use of *absmin*, precision is improved in all cases or otherwise equal. Whilst the use of *absmin* improves precision, there is no recovery applied to recover lost association

Table 4.12: Association rule precision (%) by *exact* support in *BMSWebView1*. **Min-conf = 85%**.

	ϵ_r, ϵ_c					
Noise (%)	0.8	0.75	$\frac{2}{3}$	0.5	$\frac{1}{3}$	0.25
5	**100.0**	**97.9**	93.4	39.9	47.0	46.7
10	94.5	94.4	**94.7**	62.5	57.0	55.6
15	94.2	94.1	94.2	**89.9**	73.6	73.3
20	94.5	94.3	94.3	88.2	85.4	82.7
25	93.4	93.4	93.4	86.7	**94.3**	**93.6**

rules. As such, the approximation model requires improvements to provide a more reliable recovery that is representative of a patterns true support.

4.4 Summary

Experimental results show the proposed algorithm is applicable in the real world. On one side, ARI-growth is an order of magnitude slower than RP-growth. However, a slower result to RP-growth is sensible for the following reasons: 1) RP-growth utilizes single-path optimization [TKD11], 2) RP-growths 2nd database scan only captures transactions that contain at least one rare itemset [TKD11], 3) RP-growth does not scan the initial FP-tree for each item of an itemset, and most importantly (4) ARI-growth does not incorporate the use of a strict anti-monotone property. In all cases, the proposed ARI-growth performs significantly faster than the Apriori approach. In terms of memory, large consumption is observed to be consumed by the algorithm overall. A cause of high memory consumption is the creation of apx-patterns for all generated itemsets. New apx-patterns are generated for each recursive step in the algorithm.

In the case of quality measures, improvements are needed. Few cases exist where overall quality is improved when compared with no approximation. Interesting rules are not captured reliably. When approximate support is used for rule recovery, far too many rules are lost or uninteresting to determine the true set of discriminating rules. Approximate support over-approximates to accommodate for potential loss of data and thus shifts measures for capturing an interesting rare itemset. When exact support is used in conjunction with pattern recovery, pattern recovery is more effective than the use of approximate support. Whilst recovery of interesting association rules is observed, results by the proposed technique do not show great quality improvements in the case of no applied approximation. As a result, the proposed technique cannot strictly be relied on from returned results alone with higher allowed error tolerance (e.g., $\epsilon_r = \epsilon_c = 0.5$). Additional efforts are required to ensure an interesting rule is representative of the true set of data.

Chapter 5

Conclusions

5.1 Concluding Statements

In summary, we seek to enhance the data mining field. This enhancement is with respect to the discovery of rare itemsets in noisy data. Previous works within noisy data propose lenience to the precision of the minimum support threshold that is used in frequent pattern mining. Imprecise thresholds allow for the capturing of itemsets that are interesting and frequent however, due to noise, may fall under the support threshold and otherwise be lost. Rare itemset mining has grown in popularity in recent years [BN19]. Rare itemsets are infrequent and thus when information is lost, are still infrequent, and are more sensitive to such noise. As a result, the approximation of rare itemsets with lenience to support is a difficult task. To the best of my knowledge, no work in recovering interesting rare itemsets in the presence of noise has been proposed. A definition for an approximate rare itemset is provided through modifications of frequent pattern mining recovery that apply lenience to

support.

In order to discover approximate rare itemsets in noisy data, an algorithm must be introduced. As such, the ARI-growth is a tree-based approach that generates a single RP-tree, and then captures approximate rare itemsets from the tree. ARI-growth then uses apx-patterns to evaluate if an itemset is a valid ARI. The proposed ARI-growth provides multiple improvements to recent work in approximate frequent itemsets [BL21]. First, ARI-growth does not generate all size-2 candidates from size-1 candidates. Secondly, ARI-growth constructs only a tree. Thirdly, ARI-growth takes as input mapped items ordered by frequency. With this ordering, when constructing supersets from apx-patterns, additional recursive steps are prevented by removing all supersets of items larger than the last newly added item to an itemset. Evaluation of ARI-growth is placed upon computational expense. The proposed approximation by support lenience is then tested by the quality of pattern recovery. Results show the algorithm is applicable to real-world data, performing better than an Apriori approach from traditional mining. In the case of memory consumption, ARI-growth is more constrained. Whilst ARI-growth appears applicable in cases where Apriori-inverse yields a large number of candidate itemsets, memory consumption overall is high when longer patterns are observed. In the case of pattern quality, results are unreliable with higher rates of allowed error and require further analysis to ensure there is an accurate representation of the real data. As such, the definition of an ARI may be improved in order for approximation to be more practical.

In Section 1.2.1 I ask five questions with regard to mining rare itemsets in the presence of noise by applying lenience to support. In my **M.Sc. thesis**, I provide an

answer to each question, as outlined below:

1. *How can interesting rare itemsets be discovered?* The proposed ARI-growth provides a tree-based technique to discover interesting rare itemsets. The proposed algorithm is based on a tree-based frequent pattern mining approach [BL21], adjusted to discover rare itemsets, whilst providing improvements by ignoring the generation of size-2 candidates and no longer constructing a tree for each itemset.

2. *Can itemsets that are approximated be discovered efficiently to be applicable in the real world?* Yes. While computational improvements can be made, the proposed algorithm is capable of running real-world datasets to completion.

3. *How does efficiency compare to that of traditional mining techniques?* ARI-growth outperformed Apriori-inverse in traditional mining. However, ARI-growth is more inefficient than RP-growth since approximate mining is a computationally more expensive problem.

4. *Can approximation be applied to recover lost, yet interesting, rare itemsets?* Yes, an approximation can be applied to discover rare itemsets. However, during experimentation, approximation results in insignificant and inconsistent overall improvement. Thus, current approximation models are applicable as a tool to aid in potential pattern recovery, as opposed to being relied upon as a true set of discovered patterns.

5. *Can the same approximation of itemsets be applied to recover lost, yet interesting, association rules?* Yes, similar to the recovery of itemsets, association

rules are recovered. However, the current model is insufficient for recovering association rules reliably. During experiments, up to 16% of lost rules are observed to be recovered. Usage of the approximate support overestimates true support, and thus, provides inconsistent results in measuring interesting rules with metrics such as confidence.

5.2 Limitations and Future Work

Future work of this M.Sc. thesis includes: (1) improving computational constraints, (2) improving recovery quality (3) having measures for setting thresholds (4) having functionality to accommodate for other data mining domains. The proposed ARI-growth has a number of computational constraints. ARI-growth does not include single-path or 2nd scan optimizations when compared to RP-growth. While ARI-growth no longer constructs multiple RP-trees, the initial RP-tree, and apx-patterns restrain memory. Under the circumstance where there is insufficient memory, the algorithm will not be functional. From analysis, significantly higher memory consumption to the compared RP-growth, and varied results to Apriori-inverse are observed. Analysis shows that in terms of execution time, ARI-growth is applicable in the real world since it is faster when compared to a popular Apriori approach. However, ARI-growth is an order of magnitude slower than the RP-growth.

The proposed approach was ineffective at reaching meaningful levels of recovery. Experiments show the approximation of support provides an overestimate of the true support. Whilst this aids in recovering patterns below the defined *minsup* threshold, and thereby is a used measure in approximate frequent mining; this approach is

insufficient in the case of rare itemsets. Firstly, the quantity of recovered patterns in comparison to a non-approximated approach is small. Secondly, as a result of approximated support error, association rules could not be reliably captured. Rare itemsets by definition have low support, and thus, the recovery of the pattern itself is not always sufficient. That is, metrics that can discriminate meaningful rare patterns (e.g., confidence) must also be recovered. Thus, future work is to improve upon an estimate of the true support in the presence of noise. Additionally, in our proposed model, 5 thresholds are used: $absmin$, $minsup$, $maxsup$, ϵ_r, ϵ_c. In practice, thresholds are difficult to set. In the case of ϵ_r, ϵ_c, and $absmin$, knowledge of the amount of existing noise is useful for a more accurate approximation. However, knowledge of the amount of existing noise may not be present. Thus, another possible future direction is to automate the setting of these thresholds.

Furthermore, popular domains of data mining such as high utility, uncertain, stream, or sequential data mining are not considered. There exists an underlying assumption that a pattern can be of interest through a strict evaluation of support from all provided data. In this respect, data that is probabilistic, or has a particular weight associated with its elements, or the amount of data to analyze is theoretically infinite, or data that maintains a component of time and events that must maintain order, are all inapplicable applications of this algorithm.

Bibliography

[AIS93] Rakesh Agrawal, Tomasz Imieliński, and Arun Swami. Mining association rules between sets of items in large databases. In *Proceedings of the 1993 ACM SIGMOD international conference on Management of data*, pages 207–216, 1993.

[AS+94] Rakesh Agrawal, Ramakrishnan Srikant, et al. Fast algorithms for mining association rules. In *Proc. 20th int. conf. very large data bases, VLDB*, volume 1215, pages 487–499. Citeseer, 1994.

[BBL22] Mohamed Amin Benatia, David Baudry, and Anne Louis. Detecting counterfeit products by means of frequent pattern mining. *Journal of Ambient Intelligence and Humanized Computing*, 13(7):3683–3692, 2022.

[BL21] Shariq Bashir and Daphne Teck Ching Lai. Mining approximate frequent itemsets using pattern growth approach. *Information Technology and Control*, 50(4):627–644, 2021.

[BN19] Anindita Borah and Bhabesh Nath. Rare pattern mining: challenges and future perspectives. *Complex & Intelligent Systems*, 5(1):1–23, 2019.

[BSRD23] Markus Bertl, Mahtab Shahin, Peeter Ross, and Dirk Draheim. Finding indicator diseases of psychiatric disorders in bigdata using clustered association rule mining. In *Proceedings of the 38th ACM/SIGAPP Symposium on Applied Computing*, pages 826–833, 2023.

[BSVW99] Tom Brijs, Gilbert Swinnen, Koen Vanhoof, and Geert Wets. Using association rules for product assortment decisions: A case study. In *Knowledge Discovery and Data Mining*, pages 254–260, 1999.

[CYH08] Hong Cheng, Philip S Yu, and Jiawei Han. Approximate frequent itemset mining in the presence of random noise. In *Soft computing for knowledge discovery and data mining*, pages 363–389. Springer, 2008.

[FLG+16] Philippe Fournier-Viger, Jerry Chun-Wei Lin, Antonio Gomariz, Ted Gueniche, Azadeh Soltani, Zhihong Deng, and Hoang Thanh Lam. The SPMF open-source data mining library version 2. In *Joint European conference on machine learning and knowledge discovery in databases*, pages 36–40. Springer, 2016.

[GC20] Manoj Kumar Gupta and Pravin Chandra. A comprehensive survey of data mining. *International Journal of Information Technology*, 12(4):1243–1257, 2020.

[HPY00] Jiawei Han, Jian Pei, and Yiwen Yin. Mining frequent patterns without candidate generation. *ACM sigmod record*, 29(2):1–12, 2000.

[KR05] Yun Sing Koh and Nathan Rountree. Finding sporadic rules using apriori-

inverse. In *Pacific-Asia Conference on Knowledge Discovery and Data Mining*, pages 97–106. Springer, 2005.

[LPS+06] Jinze Liu, Susan Paulsen, Xing Sun, Wei Wang, Andrew Nobel, and Jan Prins. Mining approximate frequent itemsets in the presence of noise: Algorithm and analysis. In *Proceedings of the 2006 SIAM International Conference on Data Mining*, pages 407–418. SIAM, 2006.

[MEMP21] Marcos Martinez, Belén Escobar, Garcia-Diaz Maria-Elena, and Diego P. Pinto-Roa. Market basket analysis with association rules in the retail sector using orange. case study: Appliances sales company. *CLEI electronic journal*, 24(2), 2021.

[NLB+22] SWJ Nijman, AM Leeuwenberg, I Beekers, I Verkouter, JJL Jacobs, ML Bots, FW Asselbergs, KGM Moons, and TPA Debray. Missing data is poorly handled and reported in prediction model studies using machine learning: a literature review. *Journal of clinical epidemiology*, 142:218–229, 2022.

[PTH01] Jian Pei, Anthony KH Tung, and Jiawei Han. Fault-tolerant frequent pattern mining: Problems and challenges. *DMKD*, 1(42):270, 2001.

[TKD11] Sidney Tsang, Yun Sing Koh, and Gillian Dobbie. Rp-tree: rare pattern tree mining. In *Data Warehousing and Knowledge Discovery: 13th International Conference, DaWaK 2011, Toulouse, France, August 29-September 2, 2011. Proceedings 13*, pages 277–288. Springer, 2011.

[WHO] World Health Organization. International statistical classification of diseases and related health problems (ICD). Accessed June 30, 2023. `https://www.who.int/standards/classifications/classification-of-diseases`.

[YLL+20] Jin Yang, Yuanjie Li, Qingqing Liu, Li Li, Aozi Feng, Tianyi Wang, Shuai Zheng, Anding Xu, and Jun Lyu. Brief introduction of medical database and data mining technology in big data era. *Journal of Evidence-Based Medicine*, 13(1):57–69, 2020.

Glossary

Absolute Minimum Threshold (*absmin*) The minimum allowed support before an itemset is no longer classified as interesting. Used in the recursive step of the proposed ARI pattern growth algorithm..

Anti-monotone Property A rule of exact pattern mining. The rule states that all subsets of a frequent itemset are frequent.

Approximate Mining Discovering interesting patterns by estimating the true set of patterns under the presence of noise.

Approximately Supported Transaction Given an itemset X. A transaction approximately supports X if at least $|X| * \delta_r$ number of items of X are contained within the transaction.

Association Rule A relationship formed from logical implications of patterns from data. Represented in the form $A \implies B$, for itemsets A and B.

Caching Storing data for quicker access of later use.

Child A node is a child to another node if they are connected and are at a lower level of the tree.

Column Error A user-defined threshold. The proportion of approximately supporting transactions that are allowed for each item of an itemset to be classified as interesting.

Conditional Pattern A set of items that follow along the path of a tree from a given node to the tree root.

Core Pattern A pattern with exact support greater than a defined *absmin* threshold.

Depth The level of a tree. The topmost node is said to have a depth of 0, whilst the next lower level (e.g., child of the topmost node) has a depth of 1.

Exact Mining Terminology defined in this work, 'exact' meaning extracted information from data is kept in its whole form. In this regard, traditional frequent and rare pattern mining is exact, whereas approximate mining is not.

Frequent An itemset whose support is greater than or equal to the minimum support threshold. A term for mining commonly occurring itemsets.

Header Table An ordered set of items used in pattern-growth algorithms. Each item in this table links to a tree node.

International Classification of Diseases (ICD) A classification system defined by the World Health Organization that uses codes to represent an underlying disease or condition.

Item A singular element of interest in a given domain.

Itemset An itemset X is a set of items A where, $X = \{A_1, A_2, ..., A_n\}$, $n \geq 0$, with a logical ordering $\prec$, such that $A_z \prec A_{z+1}$.

Leaf A node with no children. I.e., A 'bottom-most' node within a tree.

Length The number of items of an itemset. For an itemset X, this can be denoted as $|X|$. For example, $X = \{A, B, C\}$ is an itemset of length 3; or $|X| = 3$.

Maximum Support Threshold ($maxsup$) The maximum required support of an itemset to be defined as rare. Used in rare pattern mining.

Minimal Rare Itemset A rare itemset with only frequent subsets.

Minimum Confidence Threshold ($Minconf$) The minimum required confidence of an association rule to be significant.

Minimum Support Threshold ($Minsup$) The minimum required support of an itemset to be defined as frequent. Used in frequent pattern mining.

Node In this referenced work, a node is a domain item that contains five components. A parent node, a connected node of the same item, a list of child nodes, a domain item representing the node, and a measure of support for that domain item within the relative tree position.

Noise Terminology defined in this work referencing information loss. Information loss may be a result of corrupt, lost, mistranslated, miss inputted data that would otherwise influence the results of traditional mining.

Parent A node is a child to another node if they are connected and are at a higher level of the tree.

Path A set of connected nodes.

Perfectly Sporadic Itemset A rare itemset that contains only rare subsets.

Perfectly Sporadic Rule An association rule of perfectly sporadic itemsets that surpasses a minimum confidence threshold.

Prefix-Join The joining of itemsets by their prefix. For example {A, B, C} and {C, D, E} yields no itemsets, whereas {C, D, E} and {C, D, G} yields {C, D, E, G} as both itemsets share the prefix {C, D}.

Rare Itemset An itemset whose support is less than the maximum support threshold.

Rare Itemset Problem Difficulty to discover rare itemsets due to their low frequency. Frequent pattern mining must set a low support threshold to discover rare itemsets, causing large amounts of excess computation and uninteresting patterns to be discovered.

Row Error A user-defined threshold. The proportion of allowed missing items for a transaction to be considered as an approximately supporting transaction of an itemset.

Subset An itemset that is fully contained by a different itemset. For example, consider the itemsets X & Y. X is a subset of Y if all items of X are also within Y. This is formally denoted as $X \subset Y$.

Superset An itemset that fully contains a different itemset. For example, consider the itemsets X & Y. Y is a superset of X if all items of X are also within Y. This is formally denoted as $Y \supset X$.

Support The number of transactions in a TDB that an itemset occurs. This is also referred to as frequency or largeness. We denote the support of an itemset X as $Sup(X)$.

Traditional Mining Referring to the more well-known form of frequent and rare pattern mining. That is, frequent pattern mining uses a minsup and discovers frequent patterns. Rare pattern mining uses maxsup, absmin, and discovers rare patterns.

Transaction A set of items.

Transactional Database (TDB) A transactional database is a list of transactions.

Tree A hierarchical representation of nodes.