國立中山大學資訊工程學系

碩士論文

Department of Computer Science and Engineering

National Sun Yat-sen University

Master's Thesis

不同限制條件下之多門檻可篩除項目集探勘

Erasable Itemset Mining Using Multiple Maximum Thresholds Under Different Constraints

研究生：張逸晨

Yi-Chen Chang

指導教授：洪宗貝 博士

Dr. Tzung-Pei Hong

中華民國 113 年 7 月

July 2024

論文審定書

國立中山大學研究生學位論文審定書

本校資訊工程學系碩士班

研究生張逸晨（學號：M093040087）所提論文

不同限制條件下之多門檻可篩除項目集探勘
Erasable Itemset Mining Using Multiple Maximum Thresholds Under Different Constraints

於中華民國 113 年 4 月 10 日經本委員會審查並舉行口試，符合碩士學位論文標準。

學位考試委員簽章：

召集人 陳俊豪 陳俊豪　　委　員 洪宗貝 洪宗貝

委　員 李淑敏 李淑敏　　委　員 蔡崇煒 蔡崇煒

委　員 黃偉銘 黃偉銘　　委　員 _____

指導教授(洪宗貝) 洪宗貝 (簽名)

誌謝

光陰似箭，碩士的求學生涯即將結束，回顧碩士班生涯，從當初下定決心考取研究所到完成論文的這段期間，一路上受到許多人的幫助。首先，我要特別感謝我的指導教授洪宗貝教授，在我缺乏撰寫論文經驗的情況下，洪教授從最初的論文題目發想到內容撰寫，總是不厭其煩地耐心指導，並適時地給予方向，甚至在周末休假時額外撥出私人時間，一字一句地帶領我修改論文。洪教授也時常關心我生活狀況，讓我感受到無比的溫暖。

論文口試期間，承蒙陳俊豪教授、李淑敏教授、蔡崇煒教授以及黃偉銘博士的諸多寶貴建議，他們的專業意見使本論文更加完善，在此向他們表達最誠摯的謝意。

除了教授們外，我也要感謝實驗室一同奮鬥努力的同學們。謝謝偉銘學長及張浩學長，在平常工作忙碌之餘，還撥冗回復我各種研究上的疑問。謝謝易勵、靖維、為騰、佳翔、云珮、謝碩、金航、清杉你們的互相照顧與鼓勵，為這乏味的研究生生活增添了許多歡樂與笑聲。謝謝雅萍、李讓、定紘、林均、子傑在許多事情上的幫忙與協助。期望大家在未來各自的道路上都能一帆風順。

最後，我要感謝這一路上支持我的家人，對於我當初的選擇沒有任何反對，在求學的路上成為我最堅強的後盾，讓我能夠心無旁騖的專心致力於學術研究，並順利取得碩士學位。謹以此論文獻給所有關心我的師長、同學、朋友以及家人們，沒有你們不會有這篇論文的誕生，謝謝你們。

摘要

近年來隨著大數據量快速成長，資料探勘技術發展得更成熟也更成功地解決許多問題，並應用在多個領域。然而多數的資料探勘演算法，只使用單一的門檻來解決問題，其缺點是那些人們感興趣的項目，並不全然都會滿足單一門檻，因為可能有多個因子影響他們，因此使用統一的門檻來評估不同的項目有時是不公平的。在本論文中，我們引入多門檻的概念來解決可篩除項目集探勘問題，針對每一種物品給予其合適門檻值，使其挖掘出來的結果能更符合現實的應用。我們針對每個項目集的門檻計算考慮不同的限制方式提出了四種方法。首先，我們先設計了一個最小限制條件的演算法，這種方法採用一種較嚴謹的限制，並且有反單調特性，可以快速地縮小搜尋空間。第二個方法為平均限制條件，其不具有反單調特性，不能使用傳統的方式做處理，而採用了邊際策略和排序封閉特性來解決沒有反單調特性的困境。接著我們將第二個演算法擴展到第三個函數限制條件，使用者能依據需求自行設定項目集的門檻計算函數，讓整個探勘問題更有彈性。第四個方式為最大限制條件，其使用樹狀結構來儲存所需資料解決重複掃描資料庫的問題，因此大幅減少了演算法的執行時間。在實驗中，我們比較了多門檻每種限制條件下在不同參數中的差異，並衡量每種演算法的效能。從實驗結果可以觀察出每種限制條件和演算法設計策略有其優點和缺點，最小限制條件的候選項目集和可篩除項目集數量最少，只需要少量的記憶體與執行時間；平均和函數限制條件需經過兩階段驗證並保存上一層的資訊，

因此會花費較多的執行時間和記憶體；而最大限制條件採用減少掃描資料庫次數的技術能有效提高效能，但額外儲存的資訊會增加記憶體使用量。

關鍵字： 資料探勘、可篩除項目集探勘、單門檻探勘、多門檻探勘、探勘限制條件

Abstract

With the exponential growth of big data in recent years, data mining technology has become more mature and has successfully solved many problems and applied in many fields. However, most data mining algorithms use a single threshold to address issues. The drawback is that the items of interest to people do not all meet a single threshold, as multiple factors may influence them. Therefore, using a uniform threshold to evaluate different items is sometimes unfair. In this paper, we introduce the concept of multiple thresholds to address the issue of erasable itemset mining, assigning appropriate threshold values to each item so that the results can better match real-world applications. We propose four methods for threshold calculation of each itemset under different constraints. First, we design an algorithm with the minimum constraints, which adopts a tighter constraint and has an anti-monotonic property, allowing the search space to be quickly reduced. The second method is the average constraint, which does not have the anti-monotone property and cannot be processed using traditional methods. Instead, it employs a bound strategy and sorted closure property to overcome the lack of anti-monotone property. We then extend the second algorithm to the third function constraint, where users can set the threshold calculation function for the itemsets based on their needs, making the entire mining problem more flexible. The fourth method is the maximum constraint, which uses a tree structure to store the

necessary information, solving the issue of repeated database scanning and significantly reducing the algorithm's execution time. In the experiments, we compared the differences under various constraints of multiple thresholds with different parameters and measured the performance of each algorithm. From the experimental results, it can be observed that each constraint and its corresponding algorithm design has its advantages and disadvantages. The minimum constraint generates the fewest candidate itemsets and erasable itemsets among them but requires less memory and execution time. The average and function constraints require a two-phase verification process and the preservation of the previous level's information, leading to higher execution time and memory usage. The maximum constraint adopts a technique of reducing the number of database scans, thus improving efficiency. However, its additional stored information increases memory usage.

Keywords: data mining, erasable itemset mining, single-threshold mining, multiple-threshold mining, mining constraint.

Contents

List of Figures

List of Tables

Chapter 1. Introduction

In the digital generation, we are confronted with vast amounts of data containing useful information and potential value. These massive datasets hide important patterns, trends, and knowledge. It is a challenging task to extract these treasures efficiently from the data. Data mining, an interdisciplinary technology, has thus been proposed to reveal latent information within data and help us catch the implicit meaning embedded in the data.

Data mining is a technology that combines statistics, mathematics, databases, and artificial intelligence. Its goal is to discover patterns, trends, and knowledge within big data. Applying distinct data analysis techniques can reveal unknown patterns, providing robust support for decision-making, prediction, and optimization. Data mining has extensive applications in business, healthcare, finance, social sciences, environmental sciences, etc. In the field of business, it is widely used in market analysis [2][3][4], customer relationship management [15][61][69], risk management [40][70][73], and other appliances, enhancing the competitiveness of enterprises.

Erasable itemset mining [12] is one of the essential mining problems with extensive applications. Due to unforeseen problems for a factory, such as insufficient funds, limited logistics transportation capacity, and insufficient storage space, this approach deals with the lack problem of some raw materials and the unfitness to produce low-

profit products. Therefore, it is necessary to decide which raw materials should not be purchased so that the company's loss can be controlled within an acceptable range.

In traditional erasable itemset mining, a single threshold is often used for evaluation. It implicitly means that every item(or material) is equally important. However, this approach is neither fair nor practical in the real world, as each item has unique attributes that must be considered, such as cost, volume, weight, and storage requirements. Even if two items have the same profit, one might be harder to acquire or store, leading to lower productivity. Therefore, using the same threshold for various items in a database may result in a rare item problem [58]. To address this issue, we introduce the concept of multiple thresholds, allowing each item to be evaluated based on its specific attributes with appropriate thresholds.

In this thesis, we propose four constraint methods for calculating multiple thresholds. However, due to the variations in properties caused by different constraints, a single algorithm cannot be used for processing. Therefore, we propose four algorithms for different constraints and finally compare the performance of each algorithm from different perspectives in the experiments.

The structure of the thesis is organized as follows. Chapter 2 provides a detailed survey of the research results and application domains of erasable itemset mining and multiple-threshold mining. Chapter 3 defines the problems associated with multiple-

thresholds erasable itemset mining and then introduces algorithms for the minimum constraint by leveraging the downward closure property. Afterward, because of the lack of the downward closure property with the average constraint, a bound strategy for an algorithm is designed and explained in Chapter 4. Additionally, the algorithm based on average constraint is extended to the function constraint. Chapter 5 delves into the challenges in mining and introduces enhanced maximum constraint algorithms employing tree structures and depth-first search strategies. The synthetic and real-life datasets are used to assess the performance of algorithms, and the results are shown in Chapter 6. At last, Chapter 7 summarizes the research findings presented in this thesis.

Chapter 2. Related Work

In this chapter, we will review erasable itemset mining and multiple threshold mining, including their motivations and development processes.

2.1 Erasable Itemset Mining

In a manufacturing factory, erasable itemset mining aims to identify combinations of materials that should not be procured, allowing a factory to control losses within an acceptable range. This problem was introduced by Deng et al. in 2009, and they proposed a method called a META algorithm [11]. The META algorithm employs an Apriori-like [1][2] approach, progressively discovering all erasable itemsets layer by layer. Using the downward closure property effectively reduces the search space of itemsets. However, to check whether itemsets are erasable itemsets, a notable drawback of this approach is the computation of gain values. This requires a full database scan each time, translating to one I/O system operation. It may lead to extended execution times. Subsequent to this, several improved algorithms have been introduced to address this issue, including VME [10], MERIT [12], MERIT+ [45], dMERIT+ [45], MEI [44], and BERM [25].

VME was the first algorithm to solve these issues, introduced by Deng et al. in 2010 [10]. It employed a list structure, storing additional information within PID_list to reduce the times of database scan to twice, resulting in faster execution times compared to META. However, each itemset had its own dedicated PID_list, leading to substantial memory usage, and Hong et al. then proposed an enhancement to VME called BERM [25]. BERM utilized bit vectors to simplify the recording of PID_list, significantly reducing memory consumption.

In 2012, Deng et al. presented a novel tree-based algorithm called MERIT [12]. It initially constructed a WPPC-tree in an FP-growth manner [16] and then calculated NC-sets of each itemset by traversing this tree. The NC-sets structure was used to reduce memory usage and enhance execution speed in the mining process. However, MERIT suffered from the drawback of losing some erasable itemsets during mining, leading to inaccurate results.

MERIT+ was subsequently introduced by Le et al. [45], building upon the original MERIT and incorporating a weighted index to solve the issue of not mining all erasable itemsets. However, the duplicate NC-sets caused increased memory usage. dMERIT+ [45] adopted a new structure, dNC_sets, and a hash table to eliminate redundant information, optimizing consumed memory and execution times. Currently, the fastest-

performing algorithm of erasable itemset mining is MEI, proposed by Le et al. in 2014 [44]. MEI employs a depth-first search strategy and the dPID_set structure.

In recent years, numerous problems and applications related to erasable itemset mining have been continuously proposed [46][67]. When a factory produces various products that are not always fixed in distinct demands. The factory may manufacture new products [22][30][48] and discontinue certain products [26][27] due to various factors. It may even merge with other factories [31], leading to distinct product production changes in a product database. Consequently, how effectively to find the erasable itemset in a database becomes an important issue. The simplest approach is to rerun the mining process with the updated database. However, this would consume a considerable amount of execution time. Therefore, considering only the products that have changed and the original product information that may be affected, along with designing algorithms, is preferable.

In traditional erasable itemset mining, all researchers implicitly assume that the quantity of each raw material used to produce products is the same across all products. However, the quantity of each raw material in real applications often varies. Based on this concept, the issue of quantitative erasable itemset mining [21] was proposed. Hong et al. solved this problem by incorporating the unit quantity of each material used in producing products into the production database and segmenting the quantity of each

material into different intervals based on their interval-type representation. After that, they used a similar META method to find erasable itemsets with quantitative information.

Not each product in a factory is produced throughout the entire year. A factory production schedule is determined based on its characteristics and sales volume. For example, some products are produced in different seasons, such as T-shirts in summer compared to sweaters in winter, based on the requirement of high profits within a short period. If all products in the entire year are considered, the T-shirts or sweaters may be mistakenly categorized as erasable itemsets in traditional erasable itemset mining. This consideration of temporal-based characteristics is referred to as temporal erasable itemset mining. Considering the sequential order of time is called sequential erasable itemset mining [23]. In 2021, Hong et al. enhanced the database by adding production time for each product. Each product in a database has start and end times, termed its lifespan. Thus, they proposed seven different temporal constraints based on the combination of material lifespans. The UTE [17] and UALB [28] approaches were introduced to handle these seven temporal constraints. To achieve better performance under specific temporal constraints, the DTE method [18] was proposed in 2022, followed by the UTLB [29] algorithm in 2023, which aimed to improve efficiency compared to the UTE [17] and UALB [28].

The concepts of multiple-threshold mining [19][20] and weighted mining [47] are similar. Both aim to provide distinct measurement standards for items with different characteristics. However, their approaches differ. Whether an itemset can be considered an erasable itemset depends on the thresholds and gains. The former allows each itemset to have distinct thresholds, while the latter results in each itemset having distinct calculated gains.

Frequent itemset mining and erasable itemset mining have always been two popular topics in data mining. The former uses frequency property to find frequent itemsets with the intersection concept. But the latter uses the occurrence property to find erasable itemsets with the union property. The meanings they pursue are different from each other. However, Hong et al. discovered a clever transformation method between them in 2019 [32][33]. They devised a method to convert the original database and threshold used in erasable itemset mining into another set of databases and thresholds in frequent itemset mining through a designed transformation. This transformed data was then employed in executing the frequent itemset mining algorithms, and the results obtained from frequent itemset mining were also the same as those derived from erasable itemset mining. This phenomenon is referred to as the duality of the two algorithms.

Manual operation is often adopted to verify the correctness of erasable itemset mining programs, especially since each programmer has a different coding ability. Such doing is time-consuming and costly. In 2022, Hong et al. proposed a lightweight testing method called metamorphic test [24] by identifying the relationship between the input and output of erasable itemset mining to verify whether the program satisfies those relationships. If the tested results of the program do not conform to those relationships, then it can be said to be incorrect for that program. Through this approach, the correctness of erasable itemset mining algorithms can be effectively verified.

Table 2-1 summarizes the classifications of relevant research in erasable itemset mining.

Table 2-1: Summary of erasable itemset mining

Problem	Algorithm	Year	Method
Original	META	2009 [11]	Apriori-base
Performance	VME	2010 [10]	PID_lists
	MERIT	2012 [12]	WPPC-tree, NC-sets
	MERIT+	2013 [45]	Weighted index, hash table
	dMERIT+	2013 [45]	dNC-set
	MEI	2014 [44]	PIDset
	BERM	2021 [25]	Bit vector
Top-k	VM	2013 [9]	PID_lists
	dVM	2014 [62]	dPID_list
Weighting	WEP	2015 [47]	WEP-tree
Merger	TEIMA	2016 [31]	FUP-base
Increment	IWEI	2016 [48]	IWEI-tree, OP-list
		2017 [30]	FUP-base
		2017 [22]	Pre-large
	IWEL	2020 [59]	List structure
Itemset constraint	MEIC	2017 [75]	MEI-base
Closed pattern	ECPat	2017 [74]	dPidset
Deletion		2018 [26]	FUP-base
		2018 [27]	Pre-large
Quantification	IERM	2019 [21]	Interval region
Duality		2019 [32]	Transformation
		2019 [33]	Transformation
Maximal pattern	PE-GenMax-EI	2019 [63]	MEI-base
	IME	2024 [8]	List structure
Temporality	UTE	2021 [17]	Two phases
	DTE	2022 [18]	Ascending order
	UALB	2022 [28]	Lower bound
	UTLB	2023 [29]	MEI-base
Multiple-thresholds		2022 [19]	Apriori-base
		2024 [20]	Sorted closure
Sequent		2022 [23]	Apriori-base
Program correctness		2022 [24]	Metamorphic test

2.2 Multiple-threshold Data Mining

In the field of retail marketing, association rule mining is used to analyze consumer behavior and identify consumption patterns. Due to the lower-price goods in a market like food or fruit, their selling quantities are often much higher than those of the high-priced ones such as electronics or furniture. Under a single threshold standard, each item in a dataset is measured by the same criterion. If the threshold is assigned too high, the mining results may not include the high-priced item. On the contrary, many redundant rules are generated if the threshold value is too low. This issue is known as the rare problem [58]. Liu et al. first proposed the concept of multiple thresholds in 1999 to solve this problem [54]. They suggested assigning an appropriate threshold to each item based on its characteristics, such as profit margins and ease of preservation. They designed an algorithm called MSapriori, based on the Apriori algorithm, but the mining process does not have the downward closure property. Wang et al. grouped items with the same threshold into bins and used an adaptive lower bound for each bin to achieve a two-phase approach [77]. Later, Lee et al. modified the threshold calculation method to ensure the downward closure property [50]. Tseng and Lin argued that the generalized association rules derived from categorizing items are more useful than the original concept of association rules [72]. They proposed the MMS_Cumulate algorithm to handle taxonomy at different levels. The above methods all require pre-

defined thresholds for each item. However, in 2017, Dahbi et al. introduced an approach that does not require pre-defined thresholds [7]. Instead, it automatically sets thresholds based on the support calculated for each level of itemsets. This method makes it impossible to predict the threshold each time, as it depends on the database content, thereby reducing human influence.

The methods mentioned above are based on Apriori, similar to erasable itemset mining, and they involve the issue of repeatedly scanning the database to calculate support values. To address this problem, many optimized multiple-threshold versions have been designed. Hu et al. first proposed the CFP-growth algorithm in 2006 [35], which is based on the FP-growth algorithm and uses a tree structure called the MIS-tree to store information from the product database. This approach reduces the number of scans to just two, allowing for the generation of the necessary candidate sets and support values. However, the excessive number of tree nodes can consume a large amount of memory. The CFP-growth++ [41] and OCFP [34] algorithms were proposed to prune and merge unused tree nodes, thus compressing the tree. The IMIS-tree algorithm uses cross rules to save transactions with common parts under the same parent node when building the tree [78], reducing its size and thereby lowering memory usage and execution time. Chen et al. applied the concept of genetic algorithms to the mining process to improve execution speed [5].

In addition to association rules, various multiple-threshold mining models have been researched in recent years. These include utility mining, which seeks high-profit combinations among items with large sales volumes [42][52][53][71], erasable itemset mining for factory management [19][20], temporal mining for identifying items with high sales in specific time intervals [37][39], sequential mining that considers the order of transactions [36], data mining incorporating fuzzy concepts [49][51], and partial periodic mining for regularly frequent items [79]. These models have achieved significant results. Multiple-threshold mining has even been applied in biology to help scientists discover stronger relationships between genes [55]. The goal of multiple-threshold mining is not only to address the rare problem but also to better reflect the property of items and itemsets in real-world scenarios.

Chapter 3. Multiple-Threshold Erasable Itemset Mining with the Minimum Constraint

In this section, we will first define the problem of multiple-threshold erasable itemset mining and propose a minimum constraint algorithm with the downward closure property.

3.1 Problem Definition of Multiple-Thresholds Erasable Itemset Mining

Erasable itemset mining is utilized in the management of manufacturing factories, where various products are produced, as illustrated in Table 3-1. In the product database, a single product consists of three fields: *PID*, Items and Profit. The *PID* (product identification) serves as a code to distinguish different products, and it can be named using a combination of letters and numbers. The Items represent the materials required to produce the product, and Profit is the earnings obtained after selling the product.

Imagine a scenario where a factory encounters challenging situations, such as a decline in financial resources or insufficient storage space for materials. This leads to the inability to procure all production raw materials, necessitating a decision on which materials to forego. Consequently, products reliant on these omitted materials cannot be manufactured, causing the factory to be unable to sell them and resulting in a decline

in profits. The challenge is to determine which materials not to purchase, thereby controlling losses within acceptable proportions for the factory. This problem is known as erasable itemset mining, and the different combinations of materials identified in this process are termed erasable itemsets. Subsequently, we will provide detailed definitions for the relevant terms associated with erasable itemset mining.

Table 3-1: An example of the product database *PD*

Product Database		
PID	Items	Profit
$Product_1$	*ABC*	200
$Product_2$	*DEF*	200
$Product_3$	*BCE*	100
$Product_4$	*ACD*	100
$Product_5$	*EF*	300
$Product_6$	*ACF*	100

Definition 1: (Multiple maximum thresholds) The user or factory presets a value between 0 and 1 according to the characteristics of each item, which is used to represent the percentage of the maximum loss in the total revenue that the user or factory can accept if the item is not restocked. The maximum threshold value of each item is expressed as λ and given an example in Table 3-2 which $\lambda(A) = 0.6$ and $\lambda(B) = 0.5$.

Table 3-2: An example of the maximum threshold of each item

Item	*A*	*B*	*C*	*D*	*E*	*F*
λ	0.6	0.5	0.4	0.7	0.3	0.8

Definition 2: (Total profit) Total profit represents the sum of all product profit. The formula is defined as follows:

$$Profit_{Total} = \sum_{\{P \in PD\}} P.Profit. \quad (1)$$

For example, in Table 3-1, the total profit value is 200 + 200 + 100 + 100 + 300 + 100 = 1000.

Definition 3: (Maximum gain threshold) Maximum gain threshold represents the maximum practical loss acceptable to the user or factory. The maximum gain threshold of itemset *A* is expressed as *MGT(A)*, and the formula is defined as follows:

$$MGT(A) = Profit_{Total} * \lambda(A). \quad (2)$$

Take itemsets $\{A\}$ and $\{B\}$ as an example. The total profit has been calculated in the previous definition. The maximum threshold of each item is given in Table 3-2. Thus, *MGT(A)* = 1000 * 0.6 = 600 and *MGT(B)* = 1000 * 0.5 = 500.

Definition 4: (Gain) When a particular material cannot be purchased or stocked, it will cause the products that need to be produced with this material to be unable to be manufactured. The losses caused by these products that cannot be manufactured are called gain. The gain of itemset $\{A\}$ is expressed as *gain(A)*, and the formula of gain is defined as follows, where *P* is the product in the product database:

$$gain(A) = \sum_{\{P|A \cap P.Items \neq \emptyset,\ P \in PD\}} P.Profit. \quad (3)$$

For example, in Table 3-1, there are three products containing item $\{A\}$, which are $product_1$, $product_4$ and $product_6$. Add up the profits of these three products, *gain*(*A*) = 200 + 100 + 100 = 400.

Definition 5: (Erasable itemset) The primary goal of algorithm execution is to find all itemsets that satisfy the erasable conditions. The erasable condition is that when the gain of an itemset is less than or equal to the maximum gain threshold, denoted as *gain* ≤ *MGT*. We call it an erasable itemset, abbreviated to *EI*.

3.2 Determine Itemset Maximum Gain Threshold Using Different Constraints

During the process of multiple-thresholds erasable itemset mining, the itemsets we aim to find are those with gains less than or equal to their respective maximum gain thresholds. However, unlike traditional single-threshold approaches where the threshold remains constant, each itemset in multiple-threshold mining has its unique maximum gain threshold. Therefore, determining the appropriate threshold for each itemset becomes crucial.

If an itemset contains only one item, we use the threshold initially set for that item. However, if the itemset contains more than one item, the calculation of the maximum gain threshold varies depending on different constraint conditions. In this thesis, we propose four different methods for determining these maximum gain thresholds.

Constraint 1: (minimum constraint) The maximum gain threshold for each itemset is obtained by taking the minimum *MGT* among all items in the itemset, also known as a tight constraint. The formula is defined as follows.

$$MGT_{min}(A) = min(MGT(i) \mid i \in A). \tag{4}$$

Constraint 2: (maximum constraint) The maximum gain threshold for each itemset is defined as the maximum *MGT* among all items in the itemset, also known as a loose constraint. The formula is defined as follows.

$$MGT_{max}(A) = max(MGT(i) \mid i \in A). \tag{5}$$

Constraint 3: (average constraint) The maximum gain threshold for each itemset is defined as the average of *MGT*s for all items in the itemset. The formula is defined as follows, where *n* represents the number of items in the itemset.

$$MGT_{avg}(A) = \frac{\sum_{\forall i \in A} MGT(i)}{n}. \tag{6}$$

Constraint 4: (function constraint) The user defines the calculation function for *MGT*, where the function's input and output can be one-to-one or many-to-one, but not one-to-many. The formula is defined as follows.

$$MGT_{fun}(A) = function(MGT(i) \mid i \in A). \tag{7}$$

For example, in Tables 3-1 and 3-2, the itemset $\{A, B\}$ contains item $\{A\}$ and item $\{B\}$. The maximum gain threshold of the itemset $\{A, B\}$ under different constraints is

calculated as $MGT_{min}(A, B) = \min(MGT(A), MGT(B)) = \min(600, 500) = 500$, $MGT_{max}(A, B) = \max(600, 500) = 600$, and $MGT_{avg}(A, B) = (600+500) / 2 = 550$.

3.3 Downward Closure Property

Downward closure property is a popular skill in data mining research. The concept was introduced by R. Agrawal et al. and practically used in the Apriori algorithm first [1][2]. It is also called anti-monotone property in some other papers [60][64][65][68]. The definition of downward closure property is that if a certain itemset is an erasable itemset, all of its subset itemsets must also be an erasable itemset. If we consider all possible combinations of itemsets, its space complexity will achieve two to the power *n*, where *n* is the number of items in the database because each item has two possible in every itemset, which is true or false. The space complexity represents that the number of candidate itemsets exponentially increased explosively as the number of items increased. It is too impracticable to do so in real life. That could waste lots of time and computation ability. Hence, the downward closure property is used to solve this problem effectively. It makes the search for erasable itemsets more efficient by pruning away supersets or candidate itemsets that are not erasable itemsets. Here is an example as follows.

For example, give an itemset $\{A, B, C\}$ is an erasable itemset that is less than or equal to the minimum gain threshold. All of its subset $\{A\}$, $\{B\}$, $\{C\}$, $\{A, B\}$, $\{B, C\}$, $\{A, C\}$ should must also be erasable itemset.

From another perspective of downward closure property, if a certain itemset is not an erasable itemset, its superset must also not be an erasable itemset. When the data mining algorithm uses this property, it only needs to check partial itemsets that are not pruned away but not all combination itemsets. Next, we discuss whether the multiple threshold erasable itemset mining with the minimum constraint has a downward closure property or not.

Lemma 1: Give two itemsets, A and B. If itemset B is a subset of itemset A ($B \subseteq A$), the gain of itemset B must be less than or equal to the gain of itemset A. That is $\text{gain}(B) \leq \text{gain}(A)$.

Proof: According to the calculation of gain in the Definition 4, the gain of itemset A and B can be expressed using the following formulas:

$$gain(A) = \sum_{\{P|A \cap P.Items \neq \emptyset,\, P \in PD\}} P.Profit,$$

and

$$gain(B) = \sum_{\{P|B \cap P.Items \neq \emptyset,\, P \in PD\}} P.Profit.$$

Since itemset B is a subset of itemset A, the number of itemset B is less than or equal to the number of itemset A, and items in itemset B also appear in itemset A. We could obtain the following formulas:

$$\{P \mid B \cap P.item \neq \emptyset, P \in PD\} \subseteq \{P \mid A \cap P.item \neq \emptyset, P \in PD\}.$$

Therefore, we can derive the following expressions:

$$gain(B) \leq gain(A).$$

Theorem 1: (Downward closure property) Give an itemset consisting of n items and the individual threshold of each item. If the n-itemset is an erasable itemset with the minimum constraint, all its subsets are also erasable itemsets with the minimum constraint.

Proof: According to Definition 5, itemset A is an erasable n-itemset if and only if it satisfies the following condition:

$$gain(A) \leq MGT_{min}(A).$$

Itemset B is any subset of itemset A. According to Lemma 1, the following inequality can be obtained:

$$gain(B) \leq gain(A).$$

Under the minimum constraint, as the number of items in the itemset increases, the maximum gain threshold will only decrease. Therefore, we can derive the following inequality:

$$MGT_{min}(A) \leq MGT_{min}(B).$$

Combining the above inequality, we can derive the following result:

$$gain(B) \leq MGT_{min}(B).$$

Thus, subset *B* is also an erasable itemset with the minimum constraint.

3.4 Multiple-Threshold Erasable Itemset Mining Algorithm with the Minimum Constraint

In this section, we explain how the algorithm works and how to apply the downward closure property to our proposed algorithm. In order to better understand the details of the algorithm, we will explain the algorithm step by step, and the corresponding pseudo-code line number is marked at the end of each step.

INPUT: A product database *PD* with *PID*, materials and profit. A set of maximum thresholds for each item and a parameter *k* to record what level is proceeding now. A parameter *k* is set as one initially.

OUTPUT: A set of erasable itemsets mined from product database *PD* with the minimum constraint.

STEP 1: Scan the whole database to calculate the total profit and the actual gain of each 1-itemset (Lines 12 to 17).

STEP 2: Calculate the maximum gain threshold for each item with the user presetting threshold (Lines 20 to 21).

STEP 3: Verify whether the gain of each 1-itemset is less than or equal to its user-specific maximum gain threshold. If the itemset conforms to the above condition, put it into the set of erasable 1-itemset (Lines 22 to 24).

STEP 4: Check whether the set of erasable *k*-itemsets is empty. If it is not empty, set *k* as *k*+1 and do STEP 4 to STEP 6. If it is empty, then jump to STEP 7 (Line 27).

STEP 5: Generate the candidate *k*-itemsets with the erasable (*k*-1) itemsets using a similar method to the Apriori algorithm (Line 30). The precise generation method is shown in the pseudo-code below.

Generate candidate *k*-itemsets for the minimum constraint (Theorem 1):

```
1.  Input: The set of erasable (k-1)-itemset EI_{k-1}
2.  Output: The set of candidate k-itemset CI_k
3.
4.  Insert into CI_k
5.  Select a.item_1, a.item_2, ..., a.item_{k-1}, b.item_{k-1}
6.  From EI_{k-1} a, EI_{k-1} b
7.  Where a.item_1 = b.item_1, a.item_2 = b.item_2, …, a.item_{k-2} = b.item_{k-2},
8.  a.item_{k-1} < b.item_{k-1} // Compare with ASCII
9.
10. For each candidate itemset c ∈ CI_k do
11.    For each (k-1)-subset s of c do
12.       If (s ∉ EI_{k-1}) then
13.          prune off c from CI_k
14.       End If
15.    End For
16. End For
```

STEP 6: Scan the database and check whether the gain of each candidate *k*-itemset is less than or equal to its maximum gain threshold, which is calculated as the lowest maximum gain threshold for those items contained in such itemset. If the itemset satisfies the above condition, put it into the set of erasable *k*-itemsets (Lines 31 to 40).

STEP 7: Jump back to STEP 4 and run STEP 4 to STEP 7 repeatedly.

STEP 8: Collect all the erasable itemsets at each level and output as the final mining result (Lines 41 and 44).

Erasable Itemset Mining Using Multiple Maximum Thresholds Algorithm: with the Minimum Constraint

1. **Input:** Product database *PD* and the set of thresholds of each item
2. **Output:** The set of erasable itemsets with the minimum constraint
3.
4. $k = 1$ // the amount of items in the itemset
5. $Profit_{Total} = 0$
6. $EI = \varnothing$ // Erasable itemset
7.
8. **For** each item *i* in the material **do**
9. $gain(i) = 0$
10. **End For**
11.
12. **For** each product *p* in the product database **do**
13. $Profit_{Total} = Profit_{Total} + p.\text{profit}$
14. **For** each item $i \in$ product *p* **do**
15. $gain(i) = gain(i) + p.\text{profit}$
16. **End For**
17. **End For**
18.
19. $EI_1 = \varnothing$ // Erasable 1-itemset
20. **For** each item *i* in the material **do**
21. $MGT(i) = Profit_{Total} \times \lambda(i)$ // Maximum gain threshold
22. **If (**$gain(i) \leq MGT(i)$**) then**
23. $EI_1 = EI_1 \cup i$
24. $EI = EI \cup i$
25. **End If**
26. **End For**
27.
28. **While** ($EI_k \neq$ **NULL**) **do**
29. k++
30. $EI_k = \varnothing$
31. CI_k = generate_candidate_k-itemset (EI_{k-1}) // Candidate k-itemset
32. **For** each candidate itemset $c \in CI_k$ **do**
33. **For** each product *p* in the database **do**
34. **If (** $c \cap p \neq$ **NULL** **) then**
35. $gain(c) = gain(c) + p.\text{profit}$
36. **End If**
37. **End For**
38. **If** ($gain(c) \leq (\min(MGT(i)) \mid \forall\, i \in c)$) then
39. $EI_k = EI_k \cup c$
40. **End If**
41. **End For**
42. $EI = EI \cup EI_k$
43. **End While**
44.
45. **Return** *EI*

However, there is an additional requirement for STEP 5. In the previous section, it was proved that the algorithm with the minimum constraint has a downward closure property. According to the downward closure property, it can be revealed that if the *k*-itemset is an erasable itemset, all of the (*k*-1)-subset must also be erasable itemsets. It is a necessary condition for any erasable *k*-itemset. In other words, if *k*-itemset wants to be an erasable itemset, all of its subsets must be (*k*-1)-erasable itemsets at least. That is why we use this way to generate candidates.

3.5 An Example

In this section, we demonstrate how the algorithm works through the following examples, where STEP 5 involves utilizing the downward closure property to generate candidate itemsets.

INPUT: The product database *PD* (Table 3-3) and the set of the maximum threshold of each item (Table 3-4).

Table 3-3: The product database *PD*

Product Database		
PID	Items	Profit
Product$_1$	*ABC*	200
Product$_2$	*CDE*	300
Product$_3$	*AF*	100
Product$_4$	*BFG*	200
Product$_5$	*ACDEG*	400
Product$_6$	*EFG*	500
Product$_7$	*DE*	100
Product$_8$	*BCFG*	200

Table 3-4: The set of the maximum threshold of each item

Item	*A*	*B*	*C*	*D*	*E*	*F*	*G*
λ	0.6	0.1	0.7	0.9	0.5	0.8	0.4

STEP 1: Make the first scan over the product database to calculate the total profit and the actual gain of each 1-itemset. The total profit is shown below, and the gain of all 1-itemsets is shown in Table 3-5.

$$Profit_{Total} = 200 + 300 + 100 + 200 + 400 + 500 + 100 + 200 = 2000$$

Table 3-5: The gain of all 1-itemsets

Item	*A*	*B*	*C*	*D*	*E*	*F*	*G*
gain	700	600	1100	800	1300	1000	1300

STEP 2: Calculate the maximum gain threshold of each item with its own specific threshold. Take itemset $\{A\}$ as an example. The maximum gain of itemset $\{A\}$ is calculated as 2000 * 0.6 = 1200. The maximum gain of each item is shown in Table 3-6.

Table 3-6: The maximum gain threshold of each item

Item	*A*	*B*	*C*	*D*	*E*	*F*	*G*
MGT	1200	200	1400	1800	1000	1600	800

STEP 3: Calculate the maximum gain threshold of each item with its own specific threshold. For each candidate itemset in Table 3-6, compare the gain of the itemset to its pre-defined maximum gain threshold. Take itemset $\{A\}$ and $\{B\}$

as an example. Since the gain of itemset {*A*} 700 is less than or equal to the maximum gain threshold of itemset {*A*} 1200, it is stored into erasable 1-itemsets. The gain of itemset {*B*} 400 is not less than or equal to the maximum gain threshold of itemset {*B*}, so it is not erasable 1-itemsets. After comparing the process, only itemset {*A*}, {*C*}, {*D*} and {*F*} satisfy the erasable itemset requirement and are left to generate candidate 2-itemsets (the gray area in Table 3-7). The result of erasable 1-itemsets mined from the product database is shown in the following Table 3-8.

Table 3-7: Candidate 1-itemsets

Itemset	*A*	*B*	*C*	*D*	*E*	*F*	*G*
MGT	1200	200	1400	1800	1000	1600	800
gain	700	600	1100	800	1300	1000	1300

Table 3-8: Erasable 1-itemsets

Itemset	*A*	*C*	*D*	*F*

STEP 4: Since the set of erasable 1-itemsets is not empty. Set *k* as two, where *k* is used to record the number of items in the itemset and what level is processing now.

STEP 5: The candidate 2-itemsets are generated from the erasable 1-itemsets in the same way in the Apriori algorithm using the downward closure property. The additional pruning step is applied to ensure all the candidate itemsets are meaningful. Take itemset {*A*, *B*} and {*A*, *C*} as examples. The itemset {*A*, *C*} contains two items, which are {*A*} and {*C*}. Since both the gain of itemset

{*A*} and the gain of itemset {*C*} are less than or equal to the maximum gain threshold. They can be used to combine as candidate itemset {*A*, *C*} whose maximum gain threshold with the minimum constraint is calculated as min(*MGT*(*A*), *MGT*(*C*)) = min(1200, 1400) = 1200. The itemset {*A*, *C*} is then qualified as a candidate-itemset. On the contrary example for another itemset {*A*, *B*}, since the gain of one of the contained items {*B*} 400 is higher than the maximum gain threshold of itemset {*B*} 200, the candidate 2-itemset {*A*, *B*} has no possibility to become an erasable itemset, thus delete {*A*, *B*} from the set of candidate itemsets. All the candidate 2-itemsets generated in this way are shown in Table 3-9.

Table 3-9: Candidate 2-itemsets

Itemset	*AC*	*AD*	*AF*	*CD*	*CF*	*DF*
MGT_{min}	1200	1200	1200	1400	1400	1600
gain	1200	1100	1600	1200	1900	1700

STEP 6: Scan the product database to calculate the gain of each candidate 2-itemset. Since the gain of candidate 2-itemsets {*A*, *C*}, {*A*, *D*} and {*C*, *D*} is less than or equal to their maximum gain threshold (the gray area in Table 3-9), they store into erasable 2-itemsets. The result of erasable 2-itemsets mined from the product database is shown in Table 3-10.

Table 3-10: Erasable 2-itemsets

Itemset	*AC*	*AD*	*CD*

STEP 7: Since the set of erasable 2-itemsets is not empty, k is set as three and continues to generate the candidate 3-itemsets. The candidate 3-itemset is generated using the set of erasable 2-itemsets in the same way in previous STEP 5.

STEP 8: The itemset $\{A, C\}$, $\{A, D\}$, and $\{C, D\}$ are the entire subset of itemset $\{A, C, D\}$. Therefore, they can be used to combine the candidate itemset $\{A, C, D\}$. All the candidate 3-itemsets generated are found as shown in Table 3-11. Calculate the gain of itemset $\{A, C, D\}$ and its maximum gain threshold separately, which gain(ACD) = 1300 and $MGT_{min}(ACD)$ = min($MGT(A)$, $MGT(C)$, $MGT(D)$) = min(1200, 1400, 1800) = 1200. After comparing and examining, there is no candidate itemset that the gain is less than or equal to its maximum gain threshold and stored into an erasable 3-itemset. The result of erasable 3-itemsets mined from the product database is shown in Table 3-12.

Table 3-11: Candidate 3-itemsets

Itemset	*ACD*
MGT_{min}	1200
gain	1300

Table 3-12: Erasable 3-itemsets

Itemset	(Empty)

STEP 9: The set of erasable 3-itemsets is empty. The algorithm generates no candidate sets and executes the last step.

STEP 10: Collect all the erasable itemsets at each level and output them as the erasable itemsets with the minimum constraint. The final mining result is shown in Table 3-13.

Table 3-13: Erasable itemsets with the minimum constraint

A
C
D
F
AC
AD
CD

Chapter 4. Multiple-Threshold Erasable Itemset Mining with the Average Constraint

In this chapter, we will first explore the issue of the lack of downward closure property in multiple-threshold mining with both maximum and average constraints. We will overcome this challenge by employing the Sorted closure property and upper bound strategy. Finally, we will extend the algorithm to the function constraints, the most generalized form of constraints.

4.1 Issue of Downward Closure Property

Downward closure property has been introduced in Section 3.3 and successfully adopted this strategy in the minimum constraint algorithm. In this section, we only discuss whether the maximum and average constraints have downward closure properties. Consider the following conditions.

For example, consider the itemset $\{A, C\}$. In Table 3-1, gain(AC) = 500. Table 3-2 shows that the maximum gain thresholds are $MGT_{max}(AC)$ = 600 and $MGT_{avg}(AC)$ = 500, respectively. Therefore, gain(AC) ≤ $MGT_{max}(AC)$ and gain(AC) ≤ $MGT_{avg}(AC)$, making it an erasable itemset under both constraints. According to the definition of downward closure, itemsets $\{A\}$ and $\{C\}$ should also be erasable itemsets. However,

the gain(C) = 500 > MGT(C) = 400 for itemset $\{C\}$, indicating it is not an erasable itemset.

Unfortunately, the above result shows that the maximum and average constraints do not have downward closure properties. That is, it is unable to design an Apriori-like algorithm for the maximum and average constraints to prune the candidate space. The algorithm is bound to use another skill to handle this. To solve this issue, we introduce two concepts, the sorted closure property and the upper bound strategy, in the subsequent section.

In the next section, we will explain that multiple threshold erasable itemset mining with max constraint has a sorted closure property. Through this property, the candidate itemset can be reduced effectively.

4.2 Sorted Closure Property

In order to solve the multiple threshold erasable itemset mining, not all constraints have downward closure properties. Liu et al. proposed a novel technique called sorted closure property to replace downward closure property and successfully applied it in frequent itemset mining with multiple minimum support [54]. In the traditional algorithm, to conform to people's intuition in reading, each item of an itemset is usually arranged according to the linguistic order or numerical order, such as itemset $\{A, B, C\}$

or $\{A_1, A_2, A_3\}$. However, items will be sorted in descending order according to the maximum threshold in the sorted closure property. Each item in the itemset must be arranged in this order. Through such changes, some useful features will be produced. These features can effectively prune off the number of candidate itemsets and reduce the time scanning the database for calculating gain and verification.

For example, in Table 3-2, each item is sorted in descending order according to their maximum threshold. The sorted result is shown below in Table 4-1. According to this order. itemset $\{D, E, F\}$ is rearranged as $\{F, D, E\}$. Although the order is different, we still treat it as the same itemset.

Table 4-1: An example of the maximum threshold of each item after sorting

Item	*F*	*D*	*A*	*B*	*C*	*E*
λ	0.8	0.7	0.6	0.5	0.4	0.3

Next, we will prove that multiple-threshold erasable itemset mining with the maximum constraint has a sorted closure property and how to effectively reduce the candidate itemset through the theorem.

Lemma 2: The itemset A consisting of k items represented as $\{item_1, item_2, \ldots, item_k\}$. Any ($k$-1)-subset of itemset A contains either $item_1$ or $item_2$ at least.

Proof: To prove this lemma, we adopt proof by contradiction. Suppose there is a (k-1)-subset without containing $item_1$ and $item_2$. Without $item_1$ and $item_2$, there are (k-2) items left. Because items in an itemset cannot appear repeatedly, (k-2) items cannot

construct (k-1)-itemset. This result derives a contradiction since the beginning assumption that there is a (k-1)-subset without containing $item_1$ and $item_2$. Due to this contradiction, we could prove that any (k-1)-subset has either $item_1$ or $item_2$ at least.

Theorem 2: Assume itemset B is an erasable 1-itemset with the maximum constraint, and itemset A is any 1-itemset sorted before itemset B. The gain of itemset B is must less than or equal to the maximum gain threshold of itemset A. That is $gain(B) \leq MGT(A)$.

Proof: The itemset B is an erasable 1-itemset with the maximum constraint if and only if it satisfies the following condition:

$$gain(B) \leq MGT(B).$$

Since itemset B is sorted after itemset A and the order is arranged from the highest to the lowest threshold, itemset B has a smaller threshold. This leads to the following inequality:

$$MGT(B) \leq MGT(A).$$

Therefore, it can be deduced that:

$$gain(B) \leq MGT(B) \leq MGT(A).$$

The following inequality can be obtained:

$$gain(B) \leq MGT(A).$$

Theorem 3: Assume itemset *A* is an erasable 2-itemset with the maximum constraint and represented as $\{item_1, item_2\}$. It must satisfy the following two conditions, $gain(item_1) \leq MGT(item_1)$ and $gain(item_2) \leq MGT(item_1)$.

Proof: The itemset *A* is an erasable 2-itemset with the maximum constraint if and only if it satisfies the following condition:

$$gain(A) \leq MGT_{max}(A).$$

According to Lemma 1, since itemset $\{item_1\}$ and $\{item_2\}$ are subsets of itemset *A,* the following two inequalities can be derived:

$$gain(item_1) \leq gain(A),$$

and

$$gain(item_2) \leq gain(A).$$

Since the $item_1$ is sorted before $item_2$, it has a larger threshold. The maximum gain threshold of itemset *A* can be calculated using the following equation:

$$MGT_{max}(A) = \max\big(MGT(item_1), MGT(item_2)\big)$$

$$= MGT(item_1).$$

Combining the above formulas, we can derive the following two inequalities:

$$gain(item_1) \leq gain(A) \leq MGT(item_1),$$

and

$$gain(item_2) \leq gain(A) \leq MGT(item_1).$$

Thus, the following two inequalities can be obtained:

$$gain(item_1) \leq MGT(item_1),$$

and

$$gain(item_2) \leq MGT(item_1).$$

Theorem 4: Assume itemset *A*, which consists of *k* items, is an erasable itemset with the maximum constraint. The (*k*-1)-subset of itemset *A*, which contains the front item of itemset *A,* must also be an erasable itemset with the maximum constraint.

Proof: Give an itemset *A* consisting of *k* items represented as {$item_1$, $item_2$, ..., $item_k$}. The itemset *B* is a (*k*-1)-subset of itemset *A* and includes $item_1$. Each item of maximum gain threshold is $MGT(item_1) \geq MGT(item_2) \geq \ldots \geq MGT(item_k)$, which is sorted in descending order. Suppose itemset *A* is an erasable itemset with the maximum constraint if and only if it satisfies the following condition:

$$gain(A) \leq MGT_{max}(A).$$

According to Lemma 1, since itemset *B* is a (*k*-1)-subset of itemset *A*, the following inequality can be obtained:

$$gain(B) \leq gain(A)$$

Since $item_1$ has the highest maximum gain threshold and subset B contains $item_1$, the following equation can be obtained under the maximum constraint:

$$MGT_{max}(A) = \max\big(MGT(item_1), MGT(item_2), \dots, MGT(item_k)\big)$$

$$= \max\big(MGT(item_1), \dots, MGT(item_{k-1})\big)$$

$$= MGT_{max}(B).$$

Summarizing the above equations, the following inequality can be derived:

$$gain(B) \leq gain(A) \leq MGT_{max}(A) = MGT_{max}(B).$$

The following inequality can be derived, which denotes that itemset B is also an erasable itemset with the maximum constraint

$$gain(B) \leq MGT_{max}(B).$$

Therefore, if an itemset is an erasable itemset with the maximum constraint, then any (k−1)-subset that contains the first item of the itemset must also be an erasable itemset.

Theorem 5: Assume itemset A consisting of k items is an erasable k-itemset with the maximum constraint, and the first item and second have identical maximum gain thresholds. All (k-1)-subset of itemset A must also be an erasable itemset with the maximum constraint.

Proof: Give an itemset A consisting of k items represented as $\{item_1, item_2, \ldots, item_k\}$, and itemset B is a (k-1)-subset of A. Each item of maximum gain threshold is $MGT(item_1) = MGT(item_2) \geq \ldots \geq MGT(item_k)$, which is sorted in descending order. Suppose A is an erasable itemset with the maximum constraint if and only if it satisfies the following condition:

$$gain(A) \leq MGT_{max}(A).$$

According to Lemma 2, subset B is an erasable (k-1)-itemset with the maximum constraint with following three conditions:

$$gain(B) \leq \max\big(MGT(item_2), MGT(item_3), \ldots, MGT(item_k)\big) \text{ (without } item_1),$$

$$gain(B) \leq max\big(MGT(item_1), MGT(item_3), \ldots, MGT(item_k)\big) \ (without\ item_2),$$

and

$$gain(B) \leq max(MGT(item_1), MGT(item_2), \ldots, MGT(item_{k-1})).$$

Since $item_1$ and $item_2$ have the same and highest maximum gain threshold, it can be deduced that the maximum gain thresholds calculated in the three conditions described above are all the same.

$$MGT_{max}(A) = max\big(MGT(item_1), MGT(item_2), \ldots, MGT(item_k)\big)$$

$$= max\big(MGT(item_2), \ldots, MGT(item_k)\big)$$

$$= max\big(MGT(item_1), MGT(item_3), \ldots, MGT(item_k)\big)$$

$$= max\big(MGT(item_1), MGT(item_2), \ldots, MGT(item_{k-1})\big)$$

$$= MGT_{max}(B).$$

According to Lemma 1, since itemset *B* is a (*k*-1)-subset of itemset *A*, the following inequality can be obtained:

$$gain(B) \leq gain(A).$$

Summarizing the above equations, the following inequality can be derived:

$$gain(B) \leq gain(A) \leq MGT_{max}(A) = MGT_{max}(B).$$

The following inequality can be obtained:

$$gain(B) \leq MGT_{max}(B).$$

Therefore, if an itemset is an erasable itemset with the maximum constraint and its first and second items have the same threshold, then any subset of this itemset must also be an erasable itemset.

4.3 Multiple-Threshold Erasable Itemsets Mining Algorithm with the Maximum Constraint

The sorted closure property is employed in the multiple-threshold erasable itemset mining with the maximum constraint. The concept was proposed by Wang [20], and the related theories and algorithms are completed in this thesis. The execution steps of the algorithm are as follows.

INPUT: A product database *PD* with *PID*, materials and profit. A set of maximum item

thresholds and a parameter *k* to record what level are proceeding now. A parameter *k* is set as one initially.

OUTPUT: A set of erasable itemsets mined from product database *PD* with the maximum constraint.

STEP 1: Sort the maximum threshold table according to the threshold value of each item in descending order (Line 8).

STEP 2: Scan the whole database to calculate the total profit and the actual gain of each 1-itemset (Lines 13 to 18).

STEP 3: Use the total profit obtained from the previous step to calculate the maximum gain threshold for each item (Lines 20 to 22).

STEP 4: Generate candidate 1-itemsets with pseudo-code below (Line 24).

Generate candidate 1-itemsets for the maximum constraint (Theorem 2):

```
1.  Input: The sorted set of the threshold of each item in descending order
2.  Output: The set of candidate 1-itemsets CI1
3.
4.  For each item i in the sorted set of thresholds do
5.    If (gain(i) ≤ MGT(i)) then
6.      CI1 = CI1 ∪ i // Candidate 1-itemset
7.      For each item j after item i in the sorted set of thresholds do
8.        If (gain(j) ≤ MGT(i)) then
9.          CI1 = CI1 ∪ j
10.       End If
11.     End For
12.     break
13.   End If
14. End For
15.
16. Return CI1
```

STEP 5: Verify whether the gain of each candidate 1-itemset is less than or equal to its user-specific maximum gain threshold. If the itemset conforms to the above condition, put it into the set of erasable 1-itemset (Lines 25 to 30).

STEP 6: Examine whether the set of erasable *k*-itemset is empty or not. If it is not empty, set *k* as *k*+1 and do STEP 6 to STEP 9. If it is empty, jump to STEP 10 (Lines 32 to 33).

STEP 7: Generating *k*-candidate itemsets involves considering two different parameters: $k = 2$ and $k \geq 3$.

Case 1: When $k = 2$, generate candidate 2-itemsets with pseudo-code below. Notice that the generation way takes candidate 1-itemsets but not erasable 1-itemsets as input to generate candidate 2-itemsets (Lines 35 to 36).

Generate candidate 2-itemsets for the maximum constraint (Theorem 3):

```
1.  Input: The sorted set of candidate 1-itemsets CI1
2.  Output: The sorted set of candidate 2-itemsets CI2
3.
4.  For each item i in candidate 1-itemsets CI1 do
5.    If (gain(i) ≤ MGT(i)) then
6.      For each item j sorted after item i in the material do
7.        If (gain(j) ≤ MGT(i)) then
8.          CI2 = CI2 ∪ itemset {i, j}
9.        End If
10.     End For
11.   End If
12. End For
13.
14. Return CI2
```

Case 2: When $k \geq 3$, use erasable (k-1)-itemsets to generate candidate *k*-

itemsets with pseudo-code below (Lines 37 to 38).

Generate candidate *k*-itemsets for the maximum constraint ($k \geq 3$):

```
1.  Input: The sorted set of erasable (k-1)-itemsets EI_{k-1}
2.  Output: The sorted set of candidate k-itemsets CI_k
3.
4.  Insert into CI_k
5.  Select a.item_1, a.item_2, …, a.item_{k-1}, b.item_{k-1}
6.  From EI_{k-1} a, EI_{k-1} b
7.  Where a.item_1 = b.item_1, a.item_2 = b.item_2, …, a.item_{k-2} = b.item_{k-2},
8.  a.item_{k-1} < b.item_{k-1}    // Compare with threshold
9.
10. Return CI_k
```

STEP 8: Pruning *k*-candidate itemsets involves considering two different parameters: $k = 2$ and $k \geq 3$.

Case 1: When $k = 2$, the candidate itemsets do not be pruned. Go to the next verified step directly.

Case 2: When $k \geq 3$, prune the candidate *k*-itemsets using pseudo-code below (Line 39).

Prune candidate *k*-itemsets for the maximum constraint ($k \geq 3$) (Theorem 4, 5):

```
1.  Input: The set of candidate k-itemsets CI_k
2.  Output: The pruned set of candidate k-itemsets CI_k
3.
4.  For each candidate itemset c ∈ CI_k do
5.     For each (k-1)-subset s of c do
6.        If (c.item_1 ∈ s) or (MGT(c.item_1) = MGT(c.item_2)) then
7.           If (s ∉ EI_{k-1}) then
8.              prune off c from CI_k
9.           End If
10.       End If
11.    End For
12. End For
13.
14. Return CI_k
```

STEP 9: Scan the database to calculate the gain of each candidate *k*-itemsets. Check whether the gain of each candidate *k*-itemset is less than or equal to its maximum gain threshold with the maximum constraint, which chooses the highest maximum gain threshold among those items contained in such itemset (Lines 42 to 51).

STEP 10: Jump back to STEP 7 and run STEP 7 to STEP 10 repeatedly.

STEP 11: Collect all the erasable itemsets that satisfy the maximum constraint from each level. Output them as the final mining result (Lines 52 and 55).

Erasable Itemset Mining Using Multiple Maximum Thresholds Algorithm: with the Maximum Constraint

```
1.  Input: Product database PD and the set of the threshold of each item
2.  Output: The set of erasable itemsets with the maximum constraint
3.
4.  k = 1  // the amount of items in the itemset
5.  Profit_Total = 0
6.  EI = ∅  // Erasable itemset
7.
8.  Sort the item in the material according to its threshold in descending order
9.  For each item i in the material do
10.    gain(i) = 0
11. End For
12.
13. For each product p in the PD do
14.    Profit_Total = Profit_Total + p.profit
15.    For each item i ∈ product p do
16.       gain(i) = gain(i) + p.profit
17.    End For
18. End For
19.
20. For each item i in the material do
21.    MGT(i) = Profit_Total × λ(i)    // Maximum gain threshold
22. End For
23.
24. CI1 = Generate_candidate_1-itemsets
25. For each candidate itemset c ∈ CI1 do
```

```
    If (gain(c) ≤ MGT(c)) then
      EI_1 = EI_1 ∪ c    // Erasable 1-itemset
      EI = EI ∪ c
    End If
  End For

  While ( EI_k ≠ NULL) do
    k++
    EI_k = ∅
    If ( k = 2 )
      CI_k = generate_candidate_2-itemset ( CI_1 )    // Candidate 2-itemset
    Else If ( k ≥ 3 )
      CI_k = generate_candidate_k-itemset ( EI_{k-1} )
      CI_k = prune_candidate_k-itemsets ( CI_k )
    End If

    For each candidate itemset c ∈ CI_k do
      For each product p in PD do
        If (c ∩ p ≠ NULL) then
        gain(c) = gain(c) + p.profit
        End If
      End For
      If (gain(i) ≤ (max(MGT(i)) | ∀ i ∈ c )) then
        EI_k = EI_k ∪ c
      End If
    End For
    EI = EI ∪ EI_k
  End While

  Return EI
```

4.4 Upper Bound Strategy of Average Constraint

In data mining, how to effectively reduce the candidate itemset has always been an important research topic. It can greatly decrease the execution time of generating and verifying. Therefore, many practical techniques have been developed, such as the downward closure property [2] and the sorted closure property [54]. However, not every research subject has these characteristics. Even no effective method can be found to

reduce the candidate itemset, such as the multiple thresholds erasable itemset mining with the average constraint discussed in this chapter. To deal with this difficulty, Liu et al. proposed a two-phase algorithm in high utility itemsets mining without downward closure property [56][57]. In the first stage, we will find a relatively loose conditional model called bound. This model must have two prerequisites. One is that the model must have specific characteristics or skills that can reduce the candidate itemset. The second is the itemset mined through this model, which must contain our desired itemset. Entering the second stage, we will regard the itemset mined in the previous phase as the candidate itemset and verify whether the required conditions are met individually. Although this two-phase method increases the number of candidate itemsets and verification time compared with the algorithm with downward closure property. It has fewer candidate itemsets and arithmetic complexity than the exhaustive combination method and has been successfully applied to many subjects, including fuzzy utility mining [76], fuzzy high utility mining [43], temporal fuzzy utility mining [38], incremental erasable itemset mining [30].

The most challenging part of this two-phase method is finding a suitable model that meets the above two requirements. If the restriction condition is set too loosely, many redundant itemsets will be generated. It is insignificant for us to reduce candidate itemsets. If the condition is too tight, the mining results from the model may not contain

the desired itemset. Next, we prove that the maximum constraint in the multiple threshold erasable itemset mining is the bound model of the average constraint.

Theorem 6: (Upper bound) Suppose that A is an itemset that satisfies the average constraint in erasable multiple threshold mining. The itemset A also satisfies the maximum constraint in erasable multiple threshold mining. Thus, $EI_{avg} \subseteq EI_{max}$.

Proof: Give an itemset A consisting of n items represented as $\{item_1, item_2, \ldots, item_k\}$, and each item of threshold is $\lambda(item_1) \geq \lambda(item_2) \geq \ldots \geq \lambda(item_k)$. According to Definition 5, itemset A is an erasable itemset with the average constraint if and only if the following is satisfied:

$$gain(A) \leq MGT_{avg}(A)$$

$$\leq avg\big(MGT(item_1), MGT(item_2), \ldots, MGT(item_k)\big)$$

$$= avg\big(\lambda(item_1), \lambda(item_2), \ldots, \lambda(item_k)\big) * Profit_{Total}$$

$$= \frac{\lambda(item_1) + \lambda(item_2) + \ldots + \lambda(item_k)}{k} * Profit_{Total}.$$

Replace the thresholds of items following the first item with $\lambda(item_1)$, where $\lambda(item_1)$ is highest threshold.

$$\leq \frac{\lambda(item_1) + \lambda(item_1) + \cdots + \lambda(item_1)}{k} * Profit_{Total}$$

$$\leq \frac{k * \lambda(item_1)}{k} * Profit_{Total}$$

$$\leq \lambda(item_1) * Profit_{Total}.$$

Since $\lambda(item_1)$ is the highest threshold, any itemset that includes $item_1$ has a threshold of $\lambda(item_1)$ with the maximum constraint.

$$\leq max(\lambda(item_1), \lambda(item_2), \dots, \lambda(item_k)) * Profit_{Total}$$

$$\leq max(MGT(item_1), MGT(item_2), \dots, MGT(item_k))$$

$$\leq MGT_{max}(A).$$

The above-proven result shows that the erasable itemset with the maximum constraint is the upper bound of the erasable itemset with the average constraint. In other words, if the itemset wants to be an erasable itemset with an average constraint, it should satisfy the maximum constraint at least. The relation diagram between the maximum and average constraint is shown in Figure 4-1.

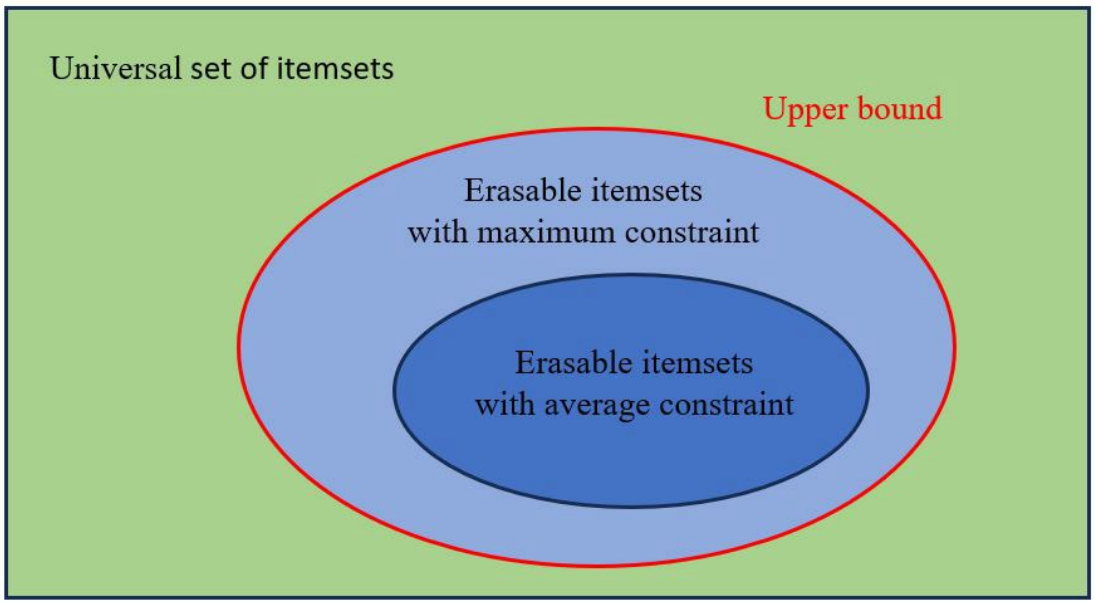

Figure 4-1: Relation diagram of the upper bound

4.5 Multiple-Threshold Erasable Itemsets Mining Algorithm with the Average Constraint

By the algorithm in Section 4.3 and the concept of upper bound, we can design the following algorithm to deal with the problem of average constraint without downward closure.

INPUT: A product database *PD* with *PID*, materials and profit. A set of maximum item thresholds and a parameter *k* to record what level are proceeding now. A parameter *k* is set as one initially.

OUTPUT: A set of erasable itemsets mined from product database *PD* with the average constraint.

STEP 1: Sort the maximum threshold table according to the threshold value of each item in descending order (Line 8).

STEP 2: Scan the whole database to calculate the total profit and the actual gain of each 1-itemset (Lines 13 to 18).

STEP 3: Use the total profit obtained from the previous step to calculate the maximum gain threshold for each item (Lines 20 to 22).

STEP 4: Look for the first 1-itemset *FI* in the sorted order in which the gain is less than or equal to the own specific maximum gain threshold. And then stored *FI* into the candidate 1-itemset (Line 24).

STEP 5: For each item sorted after the 1-itemset *FI*, verify whether the gain is less than or equal to the maximum gain threshold of *FI*. If it conforms to the above condition, put it into the candidate 1-itemset. The precise generation method of STEP 3 and STEP 4 is shown in the pseudo-code below (Line 24).

Generate candidate 1-itemsets for the average constraint (Theorem 2):

```
1.  Input: The sorted set of the threshold of each item in descending order
2.  Output: The set of candidate 1-itemsets CI1
3.
4.  For each item i in the sorted set of thresholds do
5.    If (gain(i) ≤ MGT(i)) then
6.      CI1 = CI1 ∪ i // Candidate 1-itemset
7.      For each item j sorted after item i in the set of thresholds do
8.        If (gain(j) ≤ MGT(i)) then
9.          CI1 = CI1 ∪ j
10.       End If
11.     End For
12.     break
13.   End If
14. End For
15.
16. Return CI1
```

STEP 6: Verify whether the gain of each candidate 1-itemset is less than or equal to its user-specific maximum gain threshold. If the itemset conforms to the above condition, put it into the set of erasable 1-itemset (Lines 25 to 30)

STEP 7: Examine whether the set of erasable 1-itemset is empty or not. If it is empty, end the execution of the mining algorithm and return nothing as the mining result from the product database. If it is not empty, do the next step (Lines 32 to 34).

STEP 8: Examine whether the set of candidate k-itemsets is empty or not. If it is not empty, set k as k+1 and do STEP 7 to STEP 9. If it is empty, jump to STEP 10 (Line 36).

STEP 9: In this generate candidate step, two cases need to be considered: parameters $k = 2$ and $k \geq 3$.

Case 1: When $k = 2$, the generation way takes candidate 1-itemsets but not erasable 1-itemsets as input to generate candidate 2-itemsets. Check each item i in candidate 1-itemset to see whether the gain of i is less than or equal to its maximum gain threshold. If it conforms to the above condition, then check item j sorted after item i in candidate 1-itemsets whether the gain of j is less than or equal to the maximum gain threshold of j. If item i and item j satisfy both respective conditions simultaneously, combine item i and item j as 2-itemset $\{i, j\}$ and insert it into candidate 2-itemsets. The precise generation method is shown in the pseudo-code below (Lines 39 to 40).

Generate candidate 2-itemsets for the average constraint (Theorem 3):

```
1.  Input: The sorted set of candidate 1-itemsets CI1
2.  Output: The sorted set of candidate 2-itemsets CI2
3.
4.  For each item i in candidate 1-itemsets CI1 do
5.    If (gain(i) ≤ MGT(i)) then
6.      For each item j sorted after item i in the material do
7.        If (gain(j) ≤ MGT(i)) then
8.          CI2 = CI2 ∪ itemset {i, j}
9.        End If
10.     End For
11.   End If
12. End For
13.
14. Return CI2
```

Case 2: When $k \geq 3$, pick any two itemsets out of the candidate (k-1)-itemsets, subject to the following conditions. The contained item of these two itemsets must be the same for the first (k-2), but only the item for the last (k-1) is different. The generated way is combined by putting the first (k-2) same items into the front and putting the two different last (k-1) items into the rear according to their initial sorted order. The last item has the smallest threshold. The precise generation method is shown in the pseudo-code below (Lines 41 to 42).

Generate candidate *k*-itemsets for the average constraint ($k \geq 3$):

```
1.  Input: The sorted set of candidate (k-1)-itemsets CIk-1
2.  Output: The sorted set of candidate k-itemsets CIk
3.
4.  Insert into CIk
5.  Select a.item1, a.item2, …, a.itemk-1, b.itemk-1
6.  From CIk-1 a, CIk-1 b
7.  Where a.item1 = b.item1, a.item2 = b.item2, …, a.itemk-2 = b.itemk-2,
8.  a.itemk-1 < b.itemk-1    // Compare with threshold
9.
10. Return CIk
```

STEP 10: The proof of the sorted closure property in Section 4.3 shows that the proposed pruned way can effectively prune the candidate itemset. Like the previous generation step, two cases also need to be considered.

Case 1: When $k = 2$, the candidate itemsets do not be pruned. Go to the next verified step directly.

Case 2: When $k \geq 3$, check all of the (k-1)-the subset of each candidate k-itemset c to see whether they satisfy one of the following conditions. (1) The (k-1)-subset contains the first item of the candidate k-itemset. (2) The first and second items of the k-itemset have the same maximum gain threshold. If any of the (k-1)-subset conforms to one of the above conditions, then check whether the (k-1)-subset exists in the candidate (k-1)-itemset. If the subset s does not exist in the candidate (k-1)-itemset, prune off c from candidate k-itemsets. The precise prune method is shown in the pseudo-code below (Line 43).

Prune candidate *k*-itemsets for the average constraint ($k \geq 3$) (Theorem 4, 5):

```
1.  Input: The set of candidate k-itemsets CI_k
2.  Output: The pruned set of candidate k-itemsets CI_k
3.
4.  For each candidate itemset c ∈ CI_k do
5.     For each (k-1)-subset s of c do
6.        If (c.item1 ∈ s) or (MGT(c.item1) == MGT(c.item2)) then
7.           If (s ∉ CI_k-1) then
8.              prune off c from CI_k
9.           End If
10.       End If
11.    End For
12. End For
13.
14. Return CI_k
```

STEP 11: Scan the database to calculate the gain of each candidate *k*-itemsets. Check whether the gain of each candidate *k*-itemset is less than or equal to its maximum gain threshold with the maximum constraint, which chooses the highest maximum gain threshold among those items contained in such itemset. If the candidate *k*-itemset satisfies the maximum constraint, then check whether the gain of the candidate *k*-itemset is less than or equal to its maximum gain threshold with the average constraint. If candidate *k*-itemset satisfies both conditions, store *k*-itemset into the erasable *k*-itemset. If the *k*-itemset does not conform to the maximum constraint, prune off it from candidate *k*-itemsets (Lines 46 to 59).

STEP 12: Jump back to STEP 7 and repeatedly run STEP 7 to STEP 10.

STEP 13: Collect all the erasable itemsets that satisfy the average constraint from each level. Output them as the final mining result (Lines 60 and 63).

Erasable Itemset Mining Using Multiple Maximum Thresholds Algorithm: with the Average Constraint

Input: Product database *PD* and the set of the threshold of each item
Output: The set of erasable itemsets with the average constraint

$k = 1$ // the amount of items in the itemset
$Profit_{Total} = 0$
$EI = \varnothing$ // Erasable itemset

Sort the item in the material according to its threshold in descending order
For each item *i* in the material **do**
    $gain(i) = 0$
End For

For each product *p* in the *PD* **do**
    $Profit_{Total} = Profit_{Total} + p.\text{profit}$
    For each item $i \in$ product *p* **do**
        $gain(i) = gain(i) + p.\text{profit}$
    End For
End For

For each item *i* in the material **do**
    $MGT(i) = Profit_{Total} \times \lambda(i)$ // Maximum gain threshold
End For

CI_1 = Generate_candidate_1-itemsets
For each candidate itemset $c \in CI_1$ **do**
    If ($gain(c) \leq MGT(c)$**) then**
        $EI_1 = EI_1 \cup c$ // Erasable 1-itemset
        $EI = EI \cup c$
    End If
End For

If (EI_1 = **NULL) then**
    Return Null
End If

While ($CI_k \neq$ **NULL) do**
    k++
    $EI_k = \varnothing$
    If ($k = 2$ **)**
        CI_k = generate_candidate_2-itemset (CI_1) // Candidate 2-itemset
    Else If ($k \geq 3$ **)**
        CI_k = generate_candidate_k-itemset (CI_{k-1})
        CI_k = prune_candidate_k-itemsets (CI_k)
    End If

    For each candidate itemset $c \in CI_k$ **do**
        For each product *p* in *PD* **do**

```
48.          If (c ∩ p ≠ NULL) then
49.          gain(c) = gain(c) + p.profit
50.          End If
51.        End For
52.        If (gain(i) ≤ (max(MGT(i)) | ∀ i ∈ c )) then    // Theorem 6
53.          If (gain(i) ≤ (avg(MGT(i)) | ∀ i ∈ c )) then
54.            EI_k = EI_k ∪ c
55.          End If
56.        Else
57.          prune off c from CI_k
58.        End If
59.      End For
60.      EI = EI ∪ EI_k
61.    End While
62.
63.    Return EI
```

4.6 An Example

In this section, we demonstrate how the algorithm proposed in the previous section works through the following example.

INPUT: The product database *PD* (Table 4-2) and the set of the maximum threshold of each item (Table 4-3).

Table 4-2: The product database *PD*

Product Database		
PID	Items	Profit
Product$_1$	*ABCG*	200
Product$_2$	*CDE*	300
Product$_3$	*AF*	100
Product$_4$	*AFG*	200
Product$_5$	*BG*	400
Product$_6$	*BEG*	500
Product$_7$	*DEF*	100
Product$_8$	*BDG*	200

Table 4-3: The set of the maximum threshold of each item

Item	*A*	*B*	*C*	*D*	*E*	*F*	*G*
λ	0.4	0.1	0.5	0.6	0.3	0.6	0.7

STEP 1: Sort the set of the maximum threshold in descending order. The sorted result is shown below in Table 4-4.

Table 4-4: The sorted set of the maximum threshold of each item

Item	*G*	*D*	*F*	*C*	*A*	*E*	*B*
λ	0.7	0.6	0.6	0.5	0.4	0.3	0.1

STEP 2: Make the first scan over the product database to calculate the total profit and the actual gain of each 1-itemset. The total profit is calculated below, and the gain of all 1-itemsets is shown in Table 4-5.

$$Profit_{Total} = 200 + 300 + 100 + 200 + 400 + 500 + 100 + 200 = 2000$$

Table 4-5: The gain of all 1-itemsets

Item	*G*	*D*	*F*	*C*	*A*	*E*	*B*
gain	1500	600	400	500	500	900	1300

STEP 3: Calculate the maximum gain threshold of each item with its own specific threshold using the definition. The maximum gain of each item is shown in Table 4-6.

Table 4-6: The maximum gain threshold of each item

Item	*G*	*D*	*F*	*C*	*A*	*E*	*B*
MGT	1400	1200	1200	1000	800	600	200

STEP 4: Verify the itemset from the sorted set one by one from the beginning to the end until to find the first matching erasable 1-itemset itemset and record its maximum gain threshold as the standard threshold for the next step. Start with the first itemset $\{G\}$, the gain of itemset $\{G\}$ 1500 is more than the maximum gain threshold of itemset $\{G\}$ 1400. It is not an erasable itemset. Then check the next itemset $\{D\}$. The gain of itemset $\{D\}$ 600 is less than the maximum gain threshold of itemset $\{D\}$ 1200. Store itemset $\{D\}$ into an erasable 1-itemset and record its maximum gain threshold value of 1200 for the next step (the gray area in Table 4-7). The result of the first match erasable 1-itemsets mined from the product database is shown in the following Table 4-7.

Table 4-7: First match erasable 1-itemsets

Itemset	*G*	*D*	*F*	*C*	*A*	*E*	*B*
MGT	1400	1200	1200	1000	800	600	200
gain	1500	600	400	500	500	900	1300

STEP 5: Replace all the maximum gain thresholds of itemset sorted after itemset $\{D\}$ with *MGT*(*D*) temporarily. The result is shown in the following Table 4-8.

Table 4-8: Replace the maximum gain thresholds with *MGT(D)*

Itemset	*D*	*F*	*C*	*A*	*E*	*B*
MGT(D)	1200	1200	1200	1200	1200	1200
gain	600	400	500	500	900	1300

STEP 6: For each itemset after itemset {*D*} in Table 4-7, compare the gain of the itemset to the maximum gain threshold of itemset {*D*}. Take itemset {*F*} and {*B*} as an example. Since the gain of itemset {*F*} 400 is less than or equal to the maximum gain threshold of itemset {*F*} 1200, it is added to the candidate 1-itemsets. The gain of itemset {*B*} 1300 is not less than or equal to the maximum gain threshold of itemset {*B*} 1200. Remove it from the candidate 1-itemsets. After comparing the process, only itemset {*D*}, {*F*}, {*C*}, {*A*} and {*E*} satisfy the candidate 1-itemset requirement. They are left to generate candidate 2-itemsets and verify whether it is an erasable 1-itemset (the gray area in Table 4-9). The result of generating candidate 1-itemsets is shown in the following Table 4-10.

Table 4-9: Verify the itemset following itemset {*D*} using *MGT(D)*

Itemset	*D*	*F*	*C*	*A*	*E*	*B*
MGT(D)	1200	1200	1200	1200	1200	1200
gain	600	400	500	500	900	1300

Table 4-10: generate candidate 1-itemsets

Itemset	*D*	*F*	*C*	*A*	*E*

STEP 7: For each candidate 1-itemset in Table 4-10, compare the gain of the itemset to its pre-defined maximum gain threshold. Take itemset {*D*} and {*E*} as an example. Since the gain of itemset {*D*} 600 is less than or equal to the

maximum gain threshold of itemset {*D*} 1200, it is stored into erasable 1-itemsets. The gain of itemset {*E*} 900 is not less than or equal to the maximum gain threshold of itemset {*E*} 600. *Remove* itemset {*B*} from candidate 1-itemset. After comparing the process, only itemset {*D*}, {*F*}, {*C*} and {*A*} satisfy the erasable itemset requirement (the gray area in Table 4-11). Candidate 1-itemsets are left to generate candidate 2-itemsets in Table 4-11. The result of erasable 1-itemsets mined from the product database is shown in the following Table 4-12.

Table 4-11: Candidate 1-itemsets

Itemset	*D*	*F*	*C*	*A*	*E*
MGT	1200	1200	1000	800	600
gain	600	400	500	500	900

Table 4-12: Erasable 1-itemsets

Itemset	*D*	*F*	*C*	*A*

STEP 8: Check that the set of erasable 1-itemsets is not empty. Continue execution to generate candidate itemset with level 2.

STEP 9: Using candidate 1-itemsets in Table 4-10 to generate candidate 2-itemsets. First, find the candidate 1-itemset whose gain is less than or equal to its maximum gain threshold. Then, use the maximum gain threshold of this itemset as the basis to find another 1-itemset whose gain is less than or equal to the basis maximum gain threshold sorted after the basic itemset. Combine

both 1-itemsets as candidate 2-itemset. For example, the gain of itemset {*D*} 1200 is less than the maximum gain threshold of itemset {*D*} 600. Record 1200 as the basis maximum gain threshold to verify itemset sorted after itemset {*D*} whether the gain is less than or equal to the basis maximum gain threshold 1200. After comparing, itemset {*F*}, {*C*}, {*A*} and {*E*} satisfy the above condition. Combine itemset {*D*} with itemset {*F*}, {*C*}, {*A*} and {*E*} as candidate 2-itemset {*D*, *F*}, {*D*, *C*}, {*D*, *A*} and {*D*, *E*}. Adopt the same way above for each candidate 1-itemset in Table 4-10. All the candidate 2-itemsets generated in this way are shown in Table 4-13.

Table 4-13: Generate candidate 2-itemsets

Itemset	*DF*	*DC*	*DA*	*DE*	*FC*
	FA	*FE*	*CA*	*CE*	

STEP 10: For each candidate 2-itemset in Table 4-14, Calculate its maximum gain threshold with the maximum constraint, maximum gain threshold with the average constraint and gain separately (the gray area denotes that itemsets satisfy the maximum gain threshold). Take itemset {*D*, *C*} as an example. The maximum gain thresholds with the maximum constraint and average constraint are calculated as max(*MGT*(*D*), *MGT*(*C*)) = max(1200, 1000) = 1200, avg(*MGT*(*D*), *MGT*(*C*)) = avg(1200, 1000) = 1100. Products containing itemset {*D*} and {*C*} are $product_1$, $product_2$, $product_7$ and

product$_8$. The gain is calculated as *gain(D, C)* = 200 + 300 + 100 + 200 = 800. Then, verify whether the gain of each candidate 2-itemset is less than or equal to its maximum gain threshold with maximum and average constraints. Three cases need to be considered.

Case 1: (*gain* ≤ MGT_{max} and *gain* ≤ MGT_{avg}) Remain itemset in candidate 2-itemset and store into erasable 2-itemset. For example, the itemset {*D*, *F*} satisfies the condition that *gain(D, F)* = 900 ≤ MGT_{max} = 1200 and *gain(D, F)* = 900 ≤ MGT_{avg} = 1200.

Case 2: (*gain* ≤ MGT_{max} and *gain* > MGT_{avg}) Remain itemset in candidate 2-itemset but not store into erasable 2-itemset. For example, the itemset {*D*, *A*} satisfies the condition that *gain(D, A)* = 1100 ≤ MGT_{max} = 1200 and *gain(D, A)* = 1100 > MGT_{avg} = 1000.

Case 3: (*gain* > MGT_{max}) Notice that only verify the maximum gain threshold with the maximum constraint but the average constraint because the itemset does not pass the upper bound. Remove from the candidate 2-itemset. For example, the itemset {*C*, *E*}, *gain(C, E)* = 1100 > MGT_{max} = 1000.

After comparing and removing, the result of erasable 2-itemsets mined from the product database and the remaining candidate 2-itemset are shown in

Table 4-15 and Table 4-16.

Table 4-14: Candidate 2-itemsets

Itemset	MGT_{max}	MGT_{avg}	*gain*
DF	1200	1200	900
DC	1200	1100	800
DA	1200	1000	1100
DE	1200	900	1100
FC	1200	1100	900
FA	1200	1000	600
FE	1200	900	1200
CA	1000	900	800
CE	1000	800	1100

Table 4-15: Remaining candidate 2-itemsets after comparing

Itemset	*DF*	*DC*	*DA*	*DE*	*FC*	*FA*	*FE*	*CA*

Table 4-16: erasable 2-itemsets

Itemset	*DF*	*DC*	*FC*	*FA*	*CA*

STEP 11: Check that the set of candidate 2-itemsets is not empty. Continue execution to generate candidate itemset with level 3. Be careful not to check the erasable 2-itemset like STEP 8.

STEP 12: Using candidate 2-itemset in Table 4-15 to generate candidate 3-itemset. Find any two itemsets from candidate 2-itemset which only the last item is different. Combine both of them in sorted order. For example, the itemset $\{D, F\}$ and $\{D, C\}$ have the same item $\{D\}$ before the last item and different last item $\{F\}$ and $\{C\}$. Combine itemset $\{D, F\}$ and $\{D, C\}$ into $\{D, F, C\}$. All the candidate 3-itemsets generated in this way are shown in Table 4-17.

Table 4-17: Generate candidate 3-itemsets

Itemset	*DFC*	*DFA*	*DFE*	*DCA*	*DCE*
	DAE	*FCA*	*FCE*	*FAE*	

STEP 13: Remove invalid itemsets that do not satisfy the following conditions. For each candidate 3-itemset in Table 4-17, check its all 2-subset whether its items contain the front item of the candidate itemset or if the first item and second item of the candidate itemset have the same maximum gain threshold. If the 2-subset satisfies any above conditions, it must exist in an erasable 2-itemset in Table 4-16. For example, itemset $\{F, C, A\}$ has subset $\{F, C\}$, $\{F, A\}$ and $\{C, A\}$. Subset $\{F, C\}$ and $\{F, A\}$ have the front item $\{F\}$ of itemset $\{F, C, A\}$, and they exist in an erasable 2-itemset. Subset $\{C, A\}$ do not contain front item $\{F\}$, and first item $\{F\}$ and second item $\{C\}$ of itemset $\{F, C, A\}$ do not have the same maximum gain threshold. So do not need to check whether it exists in an erasable 2-itemset. Itemset $\{F, C, A\}$ is preserved after checking. The result of preserved candidate 3-itemsets is shown in Table 4-18.

Table 4-18: Candidate 3-itemsets after pruning

Itemset	*DFC*	*DFA*	*DFE*	*DCA*	*DCE*
	DAE	*FCA*	*FCE*	*FAE*	

STEP 14: For each candidate 3-itemset in Table 4-19, Calculate and verify its

maximum gain threshold with the maximum constraint, maximum gain threshold with the average constraint and gain separately like STEP 10 (the gray area denotes that itemsets satisfy the maximum gain threshold). The result of erasable 3-itemsets mined from the product database and the remaining candidate 3-itemset are shown in Table 4-20 and Table 4-21.

Table 4-19: Candidate 3-itemsets

Itemset	MGT_{max}	MGT_{avg}	*gain*
DFC	1200	1133.3	1100
DFA	1200	1066.6	1100
DFE	1200	1000	1400
DCA	1200	1000	1100
DCE	1200	933.3	1300
DAE	1200	866.6	1600
FCA	1200	1000	900
FCE	1200	933.3	1400
FAE	1200	866.6	1400

Table 4-20: Remaining candidate 3-itemsets after comparing

Itemset	*DFC*	*DFA*	*DCA*	*FCA*

Table 4-21: Erasable 3-itemsets

Itemset	*DFC*	*FCA*

STEP 15: Check that the set of candidate 3-itemsets is not empty. Continue execution to generate candidate itemset with level 4.

STEP 16: Using candidate 3-itemset in Table 4-20 to generate candidate 4-itemset like STEP 9. All the candidate 4-itemsets generated are shown in Table 4-22.

Table 4-22: Generate candidate 4-itemsets

Itemset	*DFCA*

STEP 17: Pruning candidate 4-itemset adopts the same way as STEP 13. For example, itemset {*D*, *F*, *C*, *A*} has 3-subset {*D*, *F*, *C*}, {*D*, *F*, *A*}, {*D*, *C*, *A*} and {*F*, *C*, *A*}. The first item {*D*} and second item {*F*} of itemset {*D*, *F*, *C*, *A*} have the same maximum gain threshold. All of the 3-subsets must exist in the candidate 3-itemset. Itemset {*D*, *F*, *C*, *A*} is preserved after checking. The result of preserved candidate 4-itemsets is shown in Table 4-23.

Table 4-23: Candidate 4-itemsets after pruning

Itemset	*DFCA*

STEP 18: Verify each candidate 4-itemset in Table 4-24, similar to STEP 10. The result of erasable 3-itemsets mined from the product database and the remaining candidate 3-itemsets are shown in Table 4-25 and Table 4-26.

Table 4-24: Candidate 4-itemsets

Itemset	MGT_{max}	MGT_{avg}	*gain*
DFCA	1200	1050	1100

Table 4-25: Remaining candidate 4-itemsets after comparing

Itemset	*DFCA*

Table 4-26: Erasable 4-itemsets

Itemset	(Empty)

STEP 19: Check that the set of candidate 4-itemsets is not empty. Continue execution to generate candidate itemset with level 5.

STEP 20: Only one itemset in candidate 4-itemset in Table 4-25. It cannot generate the candidate 5-itemset in Table 4-27 because two itemsets are required.

Table 4-27: Generate candidate 5-itemsets

Itemset	(Empty)

STEP 21: There is no itemset in candidate 5-itemset in Table 4-27. Actions that do not require verification and deletion.

STEP 22: Check that the set of candidate 5-itemsets is empty. Stop generating the candidate itemset and execute the final output step.

STEP 23: Collect all the erasable itemsets at each level and output them as the erasable itemsets with the average constraint. The final mining result is shown in Table 4-28.

Table 4-28: Erasable itemsets with the average constraint

D
F
C
A
DF
DC
FC
FA
CA
DFC
FCA

4.7 Multiple-Threshold Erasable Itemsets Mining with the Function Constraint

In fact, this two-phase approach can be applied not only to average constraints but also to function constraints. We extend the algorithm to support the function constraints, allowing users to customize the calculation of multiple thresholds according to their specific needs. The average method then becomes a specific case within the function constraint framework.

The maximum gain threshold is only defined between zero and the total profit. When the maximum gain threshold exceeds the total profit, the validated itemsets will always be mined. Conversely, when it's less than zero, no itemset will meet the criteria, rendering both scenarios meaningless for verification. However, when users set the function, they often overlook all possible outcomes of its calculation, leading to values outside the defined range. For example, in Table 4-2 and Table 4-6, if the total profit is 2000 and the function is set to sum all items' maximum gain thresholds, the maximum gain threshold of itemset $\{D, G\}$ is 2400, exceeding the total profit. Hence, when setting the function, an additional parameter λ_h, representing the highest allowable loss percentage, needs to be manually provided. The MGT_h is the highest actual loss, and its function will be constrained within the following range.

$$MGT_{fun}(A) = \begin{cases} MGT_h , & f(A) > MGT_h \\ f(A) , & 0 \leq f(A) \leq MGT_h \\ 0 , & f(A) < 0 \end{cases} \quad (8)$$

MGT_h has a downward closure property similar to the META algorithm [11]. Combined with the two-phase approach, the following algorithm can be proposed.

INPUT: A product database *PD* with *PID*, materials and profit. A set of maximum item thresholds, the highest maximum threshold λ_h and a parameter *k* to record what level is proceeding now. A parameter *k* is set as one initially.

OUTPUT: A set of erasable itemsets mined from product database *PD* with the function constraint.

STEP 1: Scan the whole database to calculate the total profit and the actual gain of each item (Lines 12 to 17).

STEP 2: Use the total profit obtained from the previous step to calculate the maximum gain threshold for each item and the highest maximum gain threshold (Lines 19 to 22).

STEP 3: Verify whether each item is less than or equal to the MGT_h. If the condition is met, add the item to the candidate 1-itemset (Line 24).

Generate candidate 1-itemsets for the function constraint:

1. **Input:** The set of the threshold of each item
2. **Output:** The set of candidate 1-itemsets CI_1
3.
4. **For** each item i in the material **do**
5. **If** ($gain(i) \leq MGT_h$) **then**
6. $CI_1 = CI_1 \cup i$ // Candidate 1-itemset
7. **End If**
8. **End For**
9.
10. **Return** CI_1

STEP 4: Verify whether the gain of each candidate 1-itemset is less than or equal to the maximum gain threshold. If the condition is met, add it to the erasable 1-itemset (Lines 25 to 30).

STEP 5: Check whether the candidate 1-itemset is empty. If it is empty, proceed to STEP 10. If it is not empty, increment the parameter k by one and execute STEP 6 to STEP 8 (Lines 32 to 33).

STEP 6: Use the downward closure property to generate candidate k-itemsets from candidate (k-1)-itemsets (Line 35). The precise pseudocode is shown below.

Generate candidate *k*-itemsets for the function constraint ($k \geq 2$):

Input: The set of candidate (k-1)-itemsets CI_{k-1}
Output: The set of candidate k-itemsets CI_k

Insert into CI_k
Select $a.item_1, a.item_2, \ldots, a.item_{k-1}, b.item_{k-1}$
From $CI_{k-1}\ a, CI_{k-1}\ b$
Where $a.item_1 = b.item_1, a.item_2 = b.item_2, \ldots, a.item_{k-2} = b.item_{k-2},$
$a.item_{k-1} < b.item_{k-1}$ // Compare with ASCII

For each candidate itemset $c \in CI_k$ **do** // Downward closure property
  For each (k-1)-subset s of c **do**
    If ($s \notin CI_{k-1}$) **then**
      prune off c from CI_k
    End If
  End For
End For

Return CI_k

STEP 7: Scan the database for each candidate k-itemset to obtain its gain and calculate MGT_{fun} using MGT_h and a custom function. Verify whether the gain value of each candidate k-itemset is less than or equal to MGT_h and MGT_{fun}, and divide it into the following three cases (Lines 37 to 50).

Case 1: ($gain \leq MGT_h$ and $gain \leq MGT_{fun}$) Add the candidate k-itemset to the erasable k-itemset.

Case 2: ($gain \leq MGT_h$ and $gain > MGT_{fun}$) Do nothing and continue to verify the next candidate k-itemset.

Case 3: ($gain > MGT_h$) Remove the itemset from the candidate k-itemset.

STEP 8: Jump back to STEP 5 and run STEP 5 to STEP 8 repeatedly.

STEP 9: Collect all the erasable itemsets that satisfy the function constraint from each

level. Output them as the final mining result (Lines 51 and 54).

Erasable Itemset Mining Using Multiple Maximum Thresholds Algorithm: with the Function Constraint

1. **Input:** Product database *PD*, the set of the threshold of each item and highest maximum threshold λ_h.
2. **Output:** The set of erasable itemsets with the function constraint
3.
4. $k = 1$ // the amount of items in the itemset
5. $Profit_{Total} = 0$
6. $EI = \varnothing$ // Erasable itemset
7.
8. **For** each item *i* in the material **do**
9. $gain(i) = 0$
10. **End For**
11.
12. **For** each product *p* in the *PD* **do**
13. $Profit_{Total} = Profit_{Total} + p.\text{profit}$
14. **For** each item $i \in$ product *p* **do**
15. $gain(i) = gain(i) + p.\text{profit}$
16. **End For**
17. **End For**
18.
19. $MGT_h = Profit_{Total} \times \lambda_h$
20. **For** each item *i* in the material **do**
21. $MGT(i) = Profit_{Total} \times \lambda(i)$ // Maximum gain threshold of each item
22. **End For**
23.
24. CI_1 = Generate_candidate_1-itemsets
25. **For** each candidate itemset $c \in CI_1$ **do**
26. **If (**$gain(c) \leq MGT(c)$**) then**
27. $EI_1 = EI_1 \cup i$ // Erasable 1-itemset
28. $EI = EI \cup i$
29. **End If**
30. **End For**
31.
32. **While** ($CI_k \neq$ **NULL**) **do**
33. k++
34. $EI_k = \varnothing$
35. CI_k = generate_candidate_*k*-itemset (CI_{k-1})
36.
37. **For** each candidate itemset $c \in CI_k$ **do**
38. **For** each product *p* in *PD* **do**
39. **If** ($c \cap p \neq$ **NULL**) **then**
40. $gain(c) = gain(c) + p.\text{profit}$
41. **End If**
42. **End For**

```
43.        If (gain(i) ≤ MGT_h) then
44.          If (gain(i) ≤ (fun(MGT(i)) | ∀ i ∈ c )) then
45.            EI_k = EI_k ∪ c
46.          End If
47.        Else
48.          prune off c from CI_k
49.        End If
50.      End For
51.      EI = EI ∪ EI_k
52.    End While
53.
54.    Return EI
```

Chapter 5. Efficient Multiple-Threshold Erasable Itemset Mining with the Maximum Constraint

In this chapter, we will first discuss the problems encountered by current mining methods and propose an approach based on MEI to improve the situation.

5.1 Dilemma and Improvement in the Apriori-based Method

In the preceding two chapters, the algorithms employ techniques such as the downward closure or sorted closure to iteratively generate and prune candidate itemsets. Subsequently, it calculates the gain value for each candidate and conducts validation. This approach is called the Apriori-based [1][2] method. However, a significant drawback of this method is the need to rescan the entire database for each computation of itemset gain, resulting in a costly I/O operation for computer systems. I/O is the most time-consuming operation, and this can lead to excessively long algorithm execution times. To solve the issue of repeated scanning, Han et al. introduced a tree structure called FP-tree [16]. This framework reduces the number of database scans to twice, significantly reducing execution time.

In the context of erasable itemset mining, a similar issue arises, prompting the development of various algorithms with different structures, such as VME [10], MERIT [12], MERIT+ [45], dMERIT+ [45] and BERM [25]. Among these, the algorithm that

currently exhibits the best performance is the MEI algorithm [44], introduced by Le and Vo in 2014. Mei algorithm employs a depth-first search strategy and tree structure, constructing nodes through recursive calls to swiftly identify all erasable itemsets.

However, the MEI method cannot be directly applied due to the lack of downward closure property under multiple-threshold erasable itemset mining with the maximum constraint. Therefore, in this chapter, we propose a new structure and algorithm based on the MEI method to accelerate the performance of multiple-threshold algorithms with the maximum constraint. We refer to this as the MEI-based approach.

The improved node structure is illustrated in Figure 5-1.

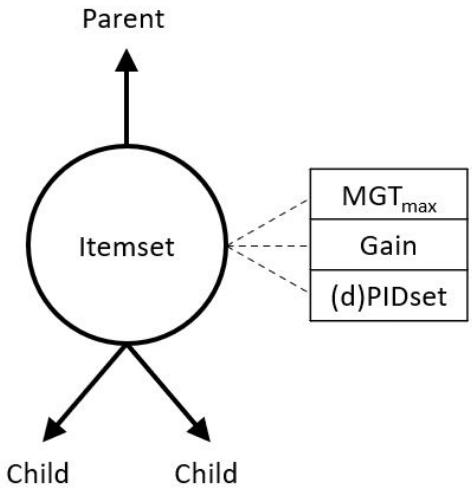

Figure 5-1: Node structure

Each node not only records the itemset but also includes three additional pieces of information: MGT_{max}, *gain*, and *PIDset*. These three pieces of information aid the

algorithm in quickly calculating gain values and determining the maximum *MGT* for the itemset. Definitions and explanations for each are provided below:

Definition 6: (*PIDset*) *PIDset* is employed to record the products in which a material is used [44]. The *PIDset* for itemset *A* is denoted as *PIDset*(*A*), and its formula is expressed as follows, where *p* represents a product in the database:

$$PIDset(A) = \{P.PID | A \cap P.Items \neq \emptyset, P \in PD\}. \tag{9}$$

Definition 7: (*dPIDset*) *dPIDset* represents the set difference between the *PIDset* of two itemsets [44]. Given two *n*-itemsets, *A* and *B*, the first (*n*-1) items are identical. When these two itemsets combine to (*n*+1)-itemset, the formula for calculating *dPIDset* is as follows:

$$dPIDset(A \cup B) = PIDset(B) - PIDset(A). \tag{10}$$

Based on these definitions, the following theorems can be derived.

Theorem 7: Given two *n*-itemsets, *A* and *B*, with the first (*n*-1) items being identical. When these two itemsets combine to (*n*+1)-itemset ($A \cup B$), the gain value can be calculated using *dPIDset*. The formula is as follows:

$$gain(A \cup B) = gain(A) + \sum_{PID \in dPIDset(A \cup B)} PID.Profit. \tag{11}$$

Proof: According to Definition 4, the gain can be calculated as follows:

$$gain(A \cup B) = \sum_{\{P|(A \cup B) \cap P.Items \neq \emptyset,\, P \in PD\}} P.Profit$$

$$= \sum_{\{P|A \cap P.Items \neq \emptyset,\, P \in PD\}} P.Profit + \sum_{\{P|B \cap P.Items \neq \emptyset,\, P \in PD\}} P.Profit - \sum_{\{P|(A \cap B) \cap P.Items \neq \emptyset,\, P \in PD\}} P.Profit$$

$$= gain(A) + \left(\sum_{PID \in PIDset(B)} PID.profit - \sum_{PID \in PIDset(A \cap B)} PID.profit \right)$$

$$= gain(A) + \sum_{PID \in PIDset(B) - PIDset(A \cap B)} PID.profit$$

$$= gain(A) + \sum_{PID \in PIDset(B) - PIDset(A)} PID.profit$$

$$= gain(A) + \sum_{PID \in dPIDset(A \cup B)} PID.profit.$$

Theorem 8: In every subtree with the root as the parent, each node within the subtree shares the same MGT_{max}.

Proof: In the same subtree, each node's itemset includes the item of the subtree's root. When the subtree's root generates candidate itemsets, it only combines with its right sibling node. Due to the descending order after sorting, the node on the right tends to have a smaller MGT_{max}. Therefore, the combination does not affect its original MGT_{max}. Consequently, each node possesses the same MGT_{max} as its root in the same subtree.

Based on the above theorems and structure, we can employ the following techniques to achieve acceleration:

- Single database scan
- Hash table for profit access
- Gain calculation with *dPIDset*
- Accessing MGT_{max} from parent node

5.2 Efficient Multiple-Threshold Erasable Itemset Mining Algorithm with the Maximum Constraint

By using the information stored in the tree structure and the derived theorems, the following algorithm can be designed to improve mining efficiency.

INPUT: A product database *PD* with product, material and profit. A set of maximum item thresholds.

OUTPUT: A set of erasable itemsets mined from product database *PD* with the maximum constraint.

STEP 1: Sort the items in descending order based on their respective maximum threshold values (Line 7).

STEP 2: Scan the entire product database once, summing up all profits. Use *PID* and profit as key-value pairs, adding them to the profit hash table. Simultaneously,

calculate the gain values for each 1-itemset and record their corresponding *PIDset* (Lines 13 to 20).

STEP 3: Calculate the maximum gain threshold for each item using the total profit obtained from the previous step (Lines 22 to 24).

STEP 4: Sequentially, in descending order, find the first itemset *FI* with a gain less than or equal to the maximum gain threshold. Add this itemset to the candidate 1-itemset, and record its maximum gain threshold (Line 26).

STEP 5: Sequentially verify whether items located after *FI* have a gain less than or equal to the maximum gain threshold recorded in the previous step. If satisfied condition, add the item to the candidate 1-itemset. The precise generation method of STEP 4 and STEP 5 is shown in the pseudo-code below (Line 26).

Generate candidate 1-itemsets for the maximum constraint (Theorem 2):

```
1.  Input: The sorted set of the threshold of each item in descending order
2.  Output: The set of candidate 1-itemsets CI1
3.
4.  For each item i in the sorted set of thresholds do
5.    If (i.gain ≤ i.MGTmax) then
6.      CI1 = CI1 ∪ i // Candidate 1-itemset
7.      For each item j sorted after item i in the set of thresholds do
8.        If (j.gain ≤ i.MGTmax) then
9.          CI1 = CI1 ∪ j
10.       End If
11.     End For
12.     break
13.   End If
14. End For
15.
16. Return CI1
```

STEP 6: Verify each candidate 1-itemset to check if its gain is less than or equal to the maximum gain threshold. If the condition is met, add it to the erasable itemset (Lines 27 to 31).

STEP 7: Check if the number of candidate 1-itemsets exceeds one. If so, treat the candidate 1-itemsets as input parameters and call the Span procedure (Lines 33 to 35).

STEP 8: Execute the Span procedure based on the following sub-steps to increase tree nodes. Accurate execution methods are presented in the pseudocode below.

1. Combine each input itemset with all its right sibling node itemsets to create candidate itemsets through a union operation (Line 7).
2. Calculate the *dPIDset* for each candidate itemset (Line 9).
3. Obtain the maximum gain threshold from the parent node (Line 10).
4. Calculate the gain value for each candidate itemset using the *dPIDset* (Lines 11 to 14).
5. Verify whether the gain of each candidate itemset is less than or equal to the maximum gain threshold. If satisfied condition, add them to the tree nodes and erasable itemset (Lines 15 to 18).
6. Check whether the number of added nodes is greater than one. If the condition is met, recursively call the "Span" procedure with these nodes as input parameters in STEP 8 (Lines 20 to 22).

7. Conclude the Span procedure and return (Line 25).

Procedure Span:

```
1.  Input: The set of previous itemsets PI
2.
3.  For i = 0 to |PI| - 2
4.     NI = ∅ // Itemset for next level
5.     For j = i + 1 to |PI| - 1
6.        CI = PI[i] ∪ PI[j]
7.        CI.gain = 0
8.        CI.dPIDset = PI[j].PIDset - PI[i].PIDset
9.        CI. MGT_max = PI[i]. MGT_max    // Theorem 8
10.       For each PID in CI.dPIDset do    // Theorem 7
11.          CI.gain = CI.gain + PH.search(PID)
12.       End For
13.       CI.gain = CI.gain + PI[i].gain
14.       If (CI.gain ≤ CI.MGT_max) then
15.          EI = EI ∪ CI
16.          NI = NI ∪ CI
17.       End If
18.    End For
19.    If (|NI| > 1)
20.       Call Procedure Span(NI)
21.    End If
22. End For
23.
24. Return
```

STEP 9: Return all erasable itemsets as the final result of the mining process (Line 37).

Efficient Erasable Itemset Mining Using Multiple Maximum Thresholds *Algorithm*:

with the Maximum Constraint

Input: Product database *PD* and the set of the threshold of each item
Output: The set of erasable itemsets with the maximum constraint

$Profit_{Total} = 0$
$EI = \varnothing$ // Erasable itemset

Sort the item in the material according to its threshold in descending order
For each item *i* in the material **do**
    i.*gain* = 0
    i.*PIDset* = ∅
End For

For each product *p* in the *PD* **do**
    PH.insert(*p*.*PID*, *p*.profit) // Profit hash table (key, value)
    $Profit_{Total} = Profit_{Total} + p.profit$
    For each item *i* ∈ product *p* **do**
        i.*gain* = *i*.*gain* + *p*.profit
        i.*PIDset* = *i*.*PIDset* ∪ *p*.*PID*
    End For
End For

For each item *i* in the material **do**
    $i.MGT_{max} = Profit_{Total} \times \lambda(i)$
End For

CI_1 = Generate_candidate_1-itemsets
For each candidate itemset $c \in CI_1$ **do**
    If ($c.gain \leq c.\ MGT_{max}$) **then**
        $EI = EI \cup c$
    End If
End For

If ($|CI_1| > 1$) **then**
    Call Span(CI_1)
End If

Return *EI*

5.3 An Example

In this chapter, we demonstrate how the algorithm works and use diagrams to illustrate how the tree spans.

INPUT: The product database *PD* (Table 5-1) and the set of the maximum threshold of each item (Table 5-2).

Table 5-1: The product database *PD*

Product Database		
PID	Items	Profit
$Product_1$	*ABC*	200
$Product_2$	*BE*	200
$Product_3$	*CD*	100
$Product_4$	*BEF*	300
$Product_5$	*ABE*	100
$Product_6$	*ACE*	100

Table 5-2: The set of the maximum threshold of each item

Item	*A*	*B*	*C*	*D*	*E*	*F*
Λ	0.5	0.3	0.4	0.4	0.6	0.2

STEP 1: Sort the set of the maximum threshold in descending order. The sorted result is shown below in Table 5-3.

Table 5-3: The sorted set of the maximum threshold of each item

Item	*E*	*A*	*C*	*D*	*B*	*F*
λ	0.6	0.5	0.4	0.4	0.3	0.3

STEP 2: Scan the product database to obtain the total profit, profit hash table, gain for 1-itemsets and *PIDset*. The calculation of total profit is 200 + 200 + 100 + 300 + 100 + 100 = 1000. The results for gain and *PIDset* for all 1-itemsets

are shown in Table 5-4. The profit hash table is shown in Table 5-5.

Table 5-4: The gain and *PIDset* of all 1-itemsets

Itemset	*E*	*A*	*C*	*D*	*B*	*F*
Gain	700	400	400	100	600	300
PIDset	2, 5, 6	1, 5, 6	1, 3, 6	3	1, 2, 4, 5	4

Table 5-5: The profit hash table

Key	Value
$Product_1$	200
$Product_2$	200
$Product_3$	100
$Product_4$	300
$Product_5$	100
$Product_6$	100

STEP 3: Calculate the maximum gain threshold for each item, as shown in Table 5-6 below.

Table 5-6: The maximum gain threshold of each item

Item	*E*	*A*	*C*	*D*	*B*	*F*
MGT	600	500	400	400	300	300

STEP 4: Following the sorting order, identify the first erasable 1-itemset. Verify the first itemset $\{E\}$: gain(E) = 700 > $MGT(E)$ = 600, which means it is not an erasable itemset. Then, verify the next itemset $\{A\}$: gain(A) = 400 ≤ $MGT(A)$ = 500, indicating it is an erasable itemset(the gray area in Table 5-7). Add itemset $\{A\}$ to the candidate 1-itemset and record $MGT(A)$.

Table 5-7: First match erasable 1-itemsets

Itemset	*E*	*A*	*C*	*D*	*B*	*F*
MGT	600	500	400	400	300	300
Gain	700	400	400	100	600	300

STEP 5: Take the recorded *MGT(A)* from the previous step as the temporary maximum gain threshold and verify the itemsets following itemset $\{A\}$. If their gain value is less than or equal to *MGT(A)*, add the itemset to the candidate 1-itemset (the gray area in Table 5-8). All candidate 1-itemsets are shown in Table 5-9.

Table 5-8: Verify the itemset following itemset $\{A\}$ using *MGT(A)*

Itemset	*A*	*C*	*D*	*B*	*F*
MGT(A)	500	500	500	500	500
Gain	400	400	100	600	300

Table 5-9: Candidate 1-itemsets

Itemset	*A*	*C*	*D*	*F*

STEP 6: Insert all candidate 1-itemsets into the tree, including their *MGT*, gain, and *PIDset*. The result after insertion is shown in Figure 5-2. Note that itemset $\{F\}$ is not an erasable itemset and is marked in gray to avoid confusion.

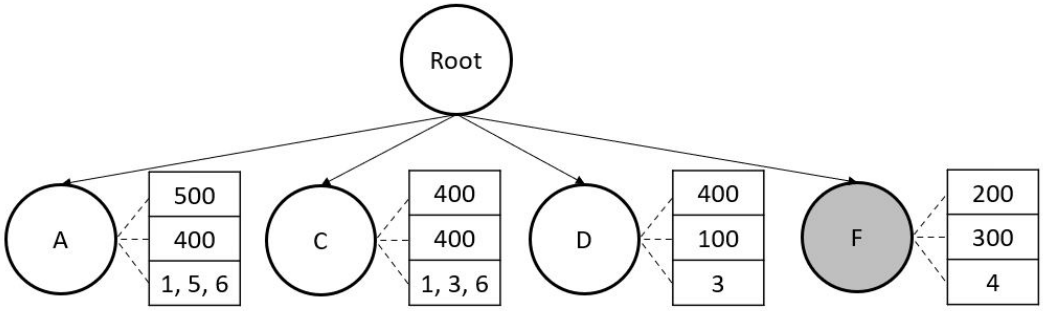

Figure 5-2**:** Tree diagram after inserting the 1-candidate itemsets

STEP 7: Check whether the number of inserted sibling nodes on the same level exceeds one. There are four candidate 1-itemsets meeting the condition, so proceed to the next level.

STEP 8: Under the depth-first search strategy, execution starts from the leftmost node itemset $\{A\}$. Itemset $\{A\}$ combines with its right sibling nodes $\{C\}$, $\{D\}$, and $\{F\}$ to generate candidate itemsets $\{A, C\}$, $\{A, D\}$, and $\{A, F\}$, respectively.

STEP 9: Each candidate itemset obtains the maximum gain threshold from the parent node itemset $\{A\}$ and calculates its *dPIDset* and gain value. Taking itemset $\{A, C\}$ as an example, $MGT_{max}(AC) = MGT(A) = 500$, $dPIDset(AC) = PIDset(C) - PIDset(A) = \{1, 3, 6\} - \{1, 5, 6\} = \{3\}$, $gain(AC) = gain(A) + dPIDset(AC).profit = 400 + 100 = 500$. After calculation, each candidate itemset is shown in Figure 5-10.

Table 5-10: The candidate itemsets information for itemset{*A*}

Itemset	*AC*	*AD*	*AF*
MGT_{max}	*500*	*500*	*500*
Gain	500	500	700
dPIDset	3	3	4

STEP 10: Verify whether the gain of the candidate itemset is less than or equal to the maximum gain threshold (the gray area in Table 5-10). Insert itemset {*A*, *C*} and {*A*, *D*} as child nodes of itemset {*A*}. The result is shown in Figure 5-3.

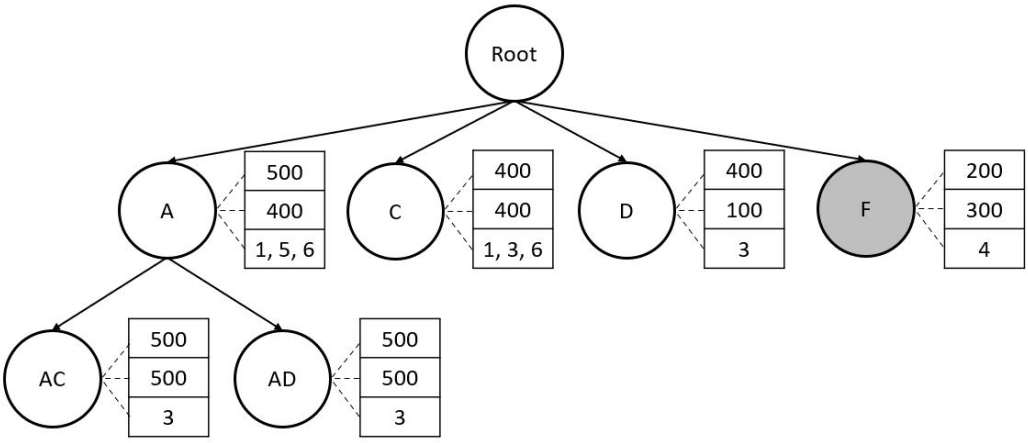

Figure 5-3: Tree diagram after inserting the child nodes for itemset {*A*}

STEP 11: Check that two nodes have been inserted, then continue to execute the next level.

STEP 12: Itemset {*A*, *C*} and its sibling node itemset {*A*, *D*} generate a new candidate itemset {*A*, *C*, *D*}. Calculate its *MGT*, *dPIDset* and gain as in STEP 9. The result is shown in Table 5-11.

Table 5-11: The candidate itemsets information for itemset{*A, C*}

Itemset	*ACD*
MGT_{max}	*500*
Gain	500
dPIDset	∅

STEP 13: Verify that itemset {*A*, *C*, *D*} is an erasable itemset. Insert it into the tree. The result after insertion is shown in Figure 5-4.

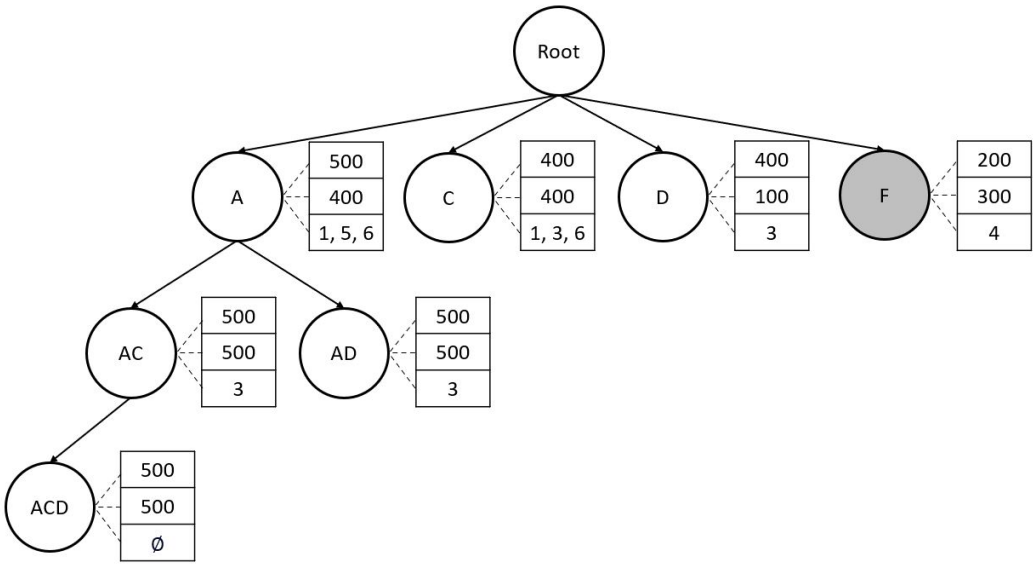

Figure 5-4: Tree diagram after inserting the child nodes for itemset {*A, C*}

STEP 14: Check that only one node has been inserted. As it does not meet the condition to proceed to the next level, return to the previous level.

STEP 15: Itemset {*A*, *D*} has no right sibling node; it cannot generate a new candidate itemset. Return to the previous level.

STEP 16: Complete each node sequentially, including itemset {*C*}, {*D*} and {*F*}. The final result is shown in Figure 5-5.

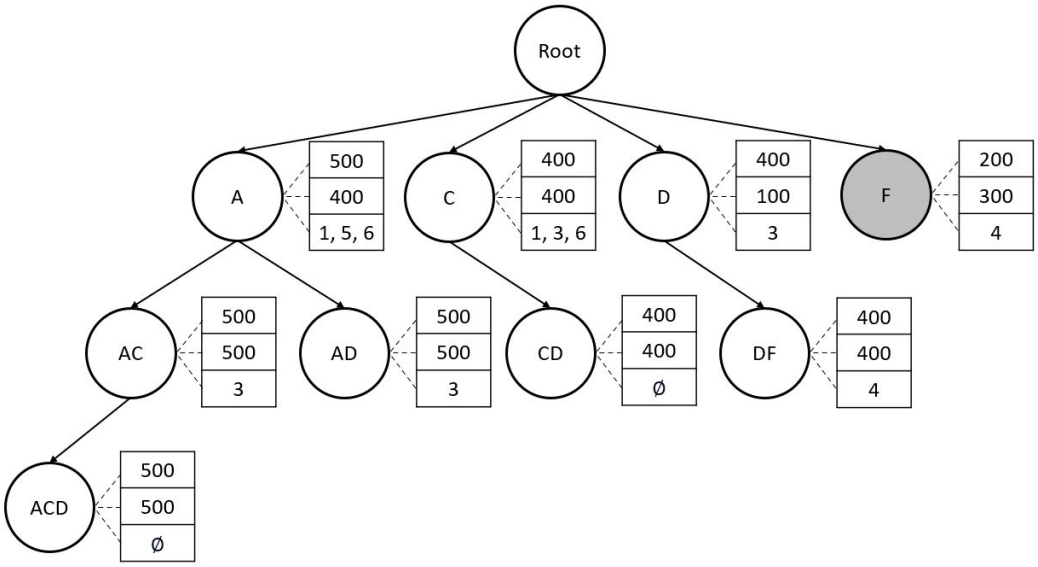

Figure 5-5: The completed mining tree diagram

STEP 17: Except for the nodes marked in gray, all other nodes are erasable itemsets in Figure 5-5. The final result is shown in Table 5-12.

Table 5-12: Erasable itemsets with the maximum constraint

A
C
D
AC
AD
CD
DF
ACD

Chapter 6. Experimental Results

In this chapter, we conduct experiments using simulated datasets and real-life datasets to evaluate the effectiveness of the algorithms proposed in the preceding chapters by varying different parameters.

6.1 Experimental Environment and Datasets

To facilitate a clearer assessment of the algorithm's performance, we maintained consistency in the experimental environment. The programs are executed on a laptop with a CPU i7-9750H@2.60GHz, 8GB RAM, running Windows 10. All programs are compiled using Java JDK 19.0.1.

Synthetic datasets used in the experiment are generated using the IBM public data generator [6][66]. These datasets are named based on the parameters used, with three parameters in total. Parameter *P* represents the quantity of products in the dataset, *I* represents the total number of items used in production, and *A* represents the average amount of items used for each product. The synthetic datasets used in the experiment are listed in Table 6-1. The density indicates the proportion of total items used for each product, calculated by dividing the average number of items per product by the total number of items. The experiment will utilize these four datasets to compare four different constraints of multiple-threshold algorithms and single-threshold algorithms.

Each parameter or threshold value will be adjusted separately while keeping other parameters constant to observe execution time, memory usage, candidate itemset, and erasable itemset quantities.

Table 6-1: Synthetic datasets

Dataset	Products	Items	Average	Density(I/A)
P100KI0.05KA10	100000	50	10	20%
PxKI0.05KA10	*x**1000	50	10	20%
P100KIxKA10	100000	*x**1000	10	20%
P100KI0.05KAx	100000	50	*x*	(*x*/50)*100%

In real-life datasets, three different datasets, namely Mushrooms, Chess, and Connect, are utilized. The parameters for each real-life dataset are shown in Table 6-2. The real-life datasets are provided by SPMF [13][14] (https://www.philippe-fournier-viger.com/spmf/index.php?link=datasets.php), an open-source Java data mining library established as a testing platform for data mining researchers. The real-life datasets have fixed sizes and cannot be adjusted. Therefore, we aimed to compare the performance between the multiple-threshold erasable itemset mining with the maximum constraint using Apriori-based and MEI-based methods across different threshold intervals to observe differences in execution time, memory usage, and candidate itemset quantities.

Table 6-2: Real-life datasets

Dataset	Products	Items	Average	Density(I/A)
Mushrooms	8416	119	23	19.33%
Chess	3196	75	37	49.33%
Connect	67557	129	43	33.33%

Both synthetic datasets generated by the IBM generator and real-life datasets collected from the real world lack a profit column, which needs to be manually added. However, the added profit is not randomly generated but follows a normal distribution $N(m, v)$, where m represents the mean and v represents the standard deviation. Most product profits are concentrated at moderate levels, with only a few products having low or high profits. This distribution better reflects real-life scenarios, where excessively high profits may lead to high product prices, and low profits may affect company operations. In all datasets, profit is generated using $N(100, 20)$.

A range interval is initially established to generate each item threshold. Multiple thresholds are then generated from this interval using a uniform distribution, denoted as $U(L, H)$, where L represents the minimum value within this threshold range, and H represents the maximum value within the threshold range.

6.2 Analysis of Multiple-Threshold Algorithms versus Single-Threshold Algorithm

In this experiment section, we will utilize synthetic datasets with four parameters: threshold interval, dataset size, number of items, and density. We will compare the differences between multiple thresholds under different constraints by adjusting only one parameter at a time. Additionally, we also compare them with the single-threshold algorithm META [11]. The single-threshold algorithm takes the highest and lowest thresholds from the threshold interval as inputs for two separate executions. In the setting of the multiple-threshold algorithm with the function constraint, we use the average as the function to compute the maximum gain threshold, and λ_h set as the highest value within the threshold interval.

6.2.1 Performance Evaluation Using Different Threshold Intervals

The experiment, involving adjustments to the threshold interval, is conducted on the P100KI0.05KA10 dataset. The threshold interval is initially set to $U(0.1, 0.2)$. With each adjustment, the entire interval is increased by 0.01 until reaching $U(0.17, 0.27)$. The experimental results are depicted in Figures 6-1 to 6-4.

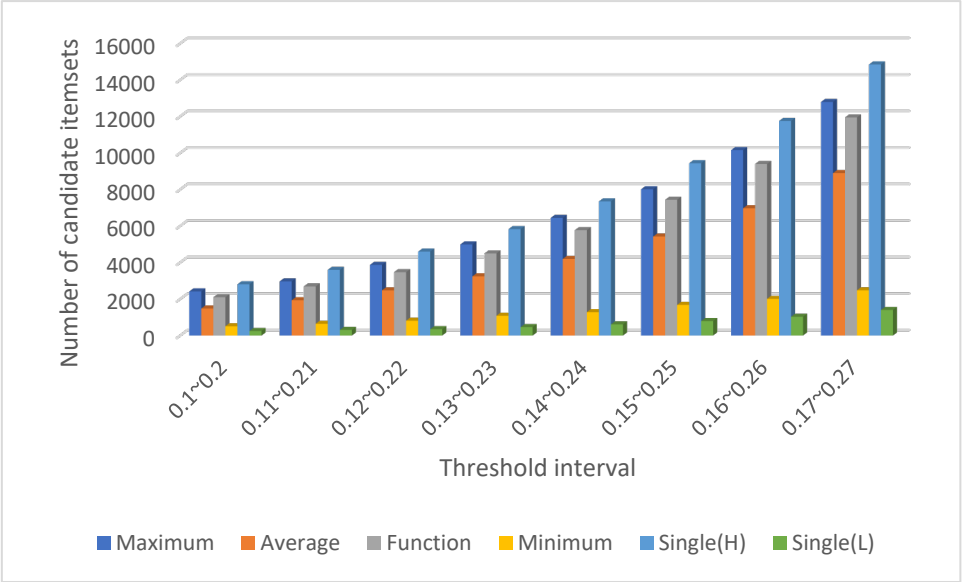

Figure 6-1: Number of candidate itemsets for different threshold intervals

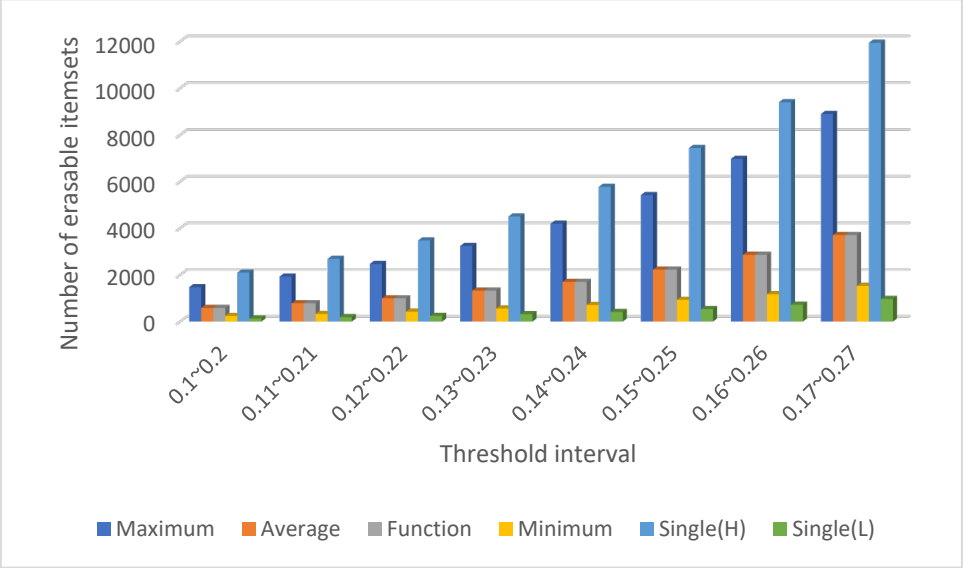

Figure 6-2: Number of erasable itemsets for different threshold intervals

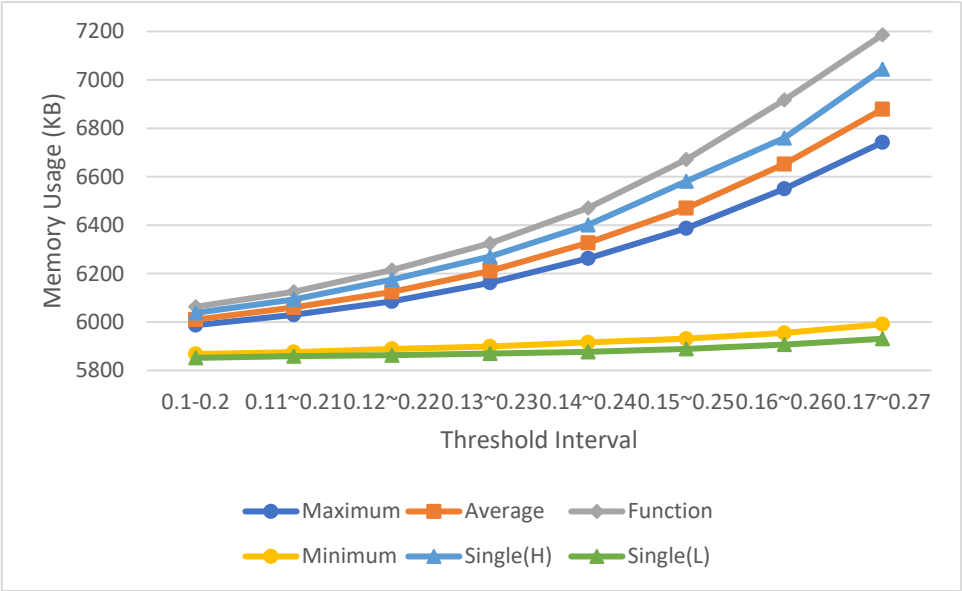

Figure 6-3: Memory usage for different threshold intervals

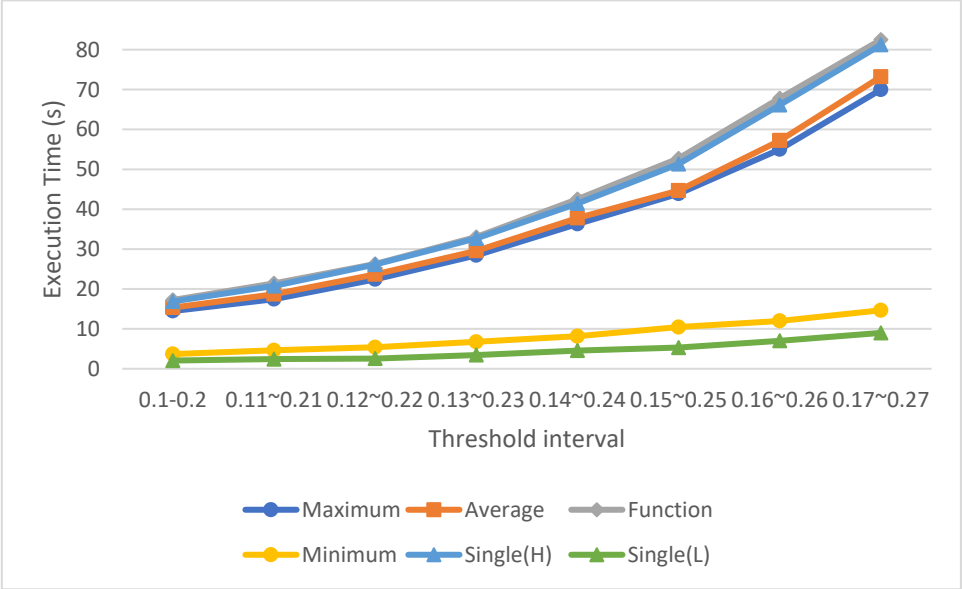

Figure 6-4: Execution time for different threshold intervals

Figures 6-1 and 6-2 show that the filtering criteria become looser as the threshold interval values increase. However, since the entire dataset remains unchanged, the gain values of each itemset also remain constant. This leads to itemsets becoming easier to meet the threshold and thus becoming erasable itemsets. Consequently, more erasable

itemsets are combined to generate more candidate itemsets. For any constraint of the multiple-thresholds algorithm, the quantity of candidate itemsets and erasable itemsets falls between the usage of the highest and lowest threshold of the threshold interval in the single-threshold approach. The maximum constraint closely resembles the single-threshold approach, employing the highest threshold of the threshold interval, while the minimum constraint approaches the usage of the lowest threshold of the threshold interval in the single-threshold approach.

As the number of candidate itemsets and erasable itemsets increases, more additional space is required for storage, leading to larger memory usage, as depicted in Figure 6-3. This not only impacts memory consumption but also requires more time for computing the gain of each itemset and verification, resulting in increased execution time, as shown in Figure 6-4.

In both the average and the function constraints, the candidate itemsets at each level must be retained for generating candidate itemsets in the next level and cannot be released. Additionally, an extra verification step is necessary to validate candidate itemsets. Therefore, memory usage and execution time will be slightly higher than the maximum constraint and single-threshold with the lowest threshold of the threshold interval as their upper bound.

Although the function constraint sets the function as average, compared to the specifically designed average constraint algorithm, it is evident that its memory usage and execution time are higher. This is because it is a design intended to cover all general cases, requiring the inclusion of more candidate itemsets to handle various situations, as shown in Figure 6-1. Figures 6-3 and 6-4 also demonstrate that its memory usage and execution time are the highest among all algorithms.

6.2.2 Performance Evaluation Using Different Dataset Sizes

The experiment adjusting the number of products is conducted on the PxKI0.05KA10 dataset. The number of products starts at 60k and increases sequentially by 20k until it reaches 200k. The threshold interval is set to $U(0.1, 0.2)$. The experimental results are depicted in Figures 6-5 to 6-8.

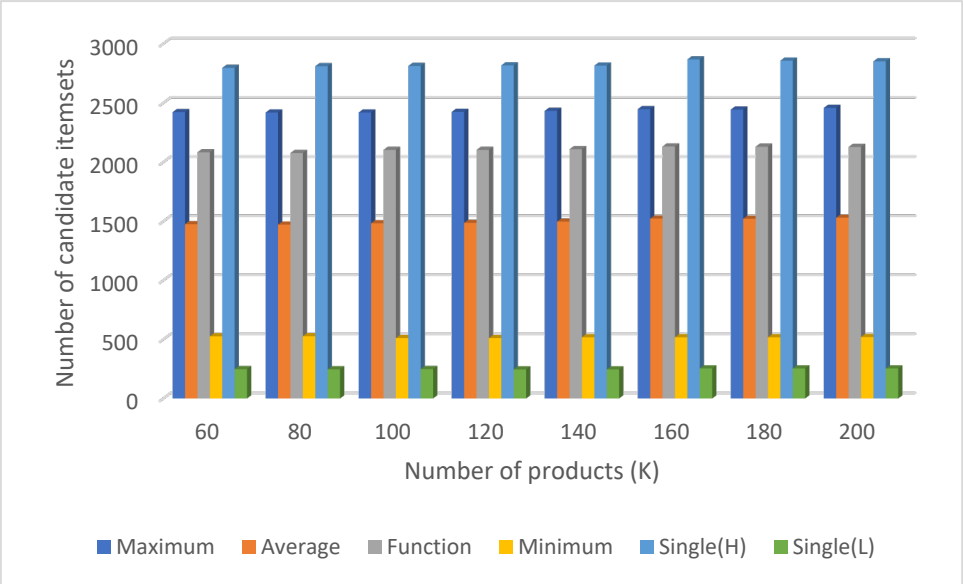

Figure 6-5: Number of candidate itemsets for different dataset sizes

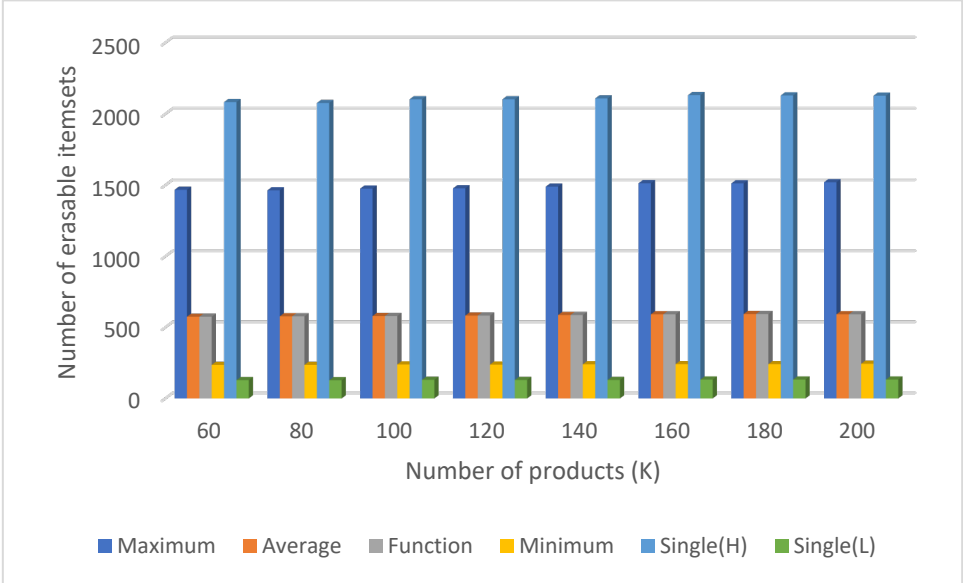

Figure 6-6: Number of erasable itemsets for different dataset sizes

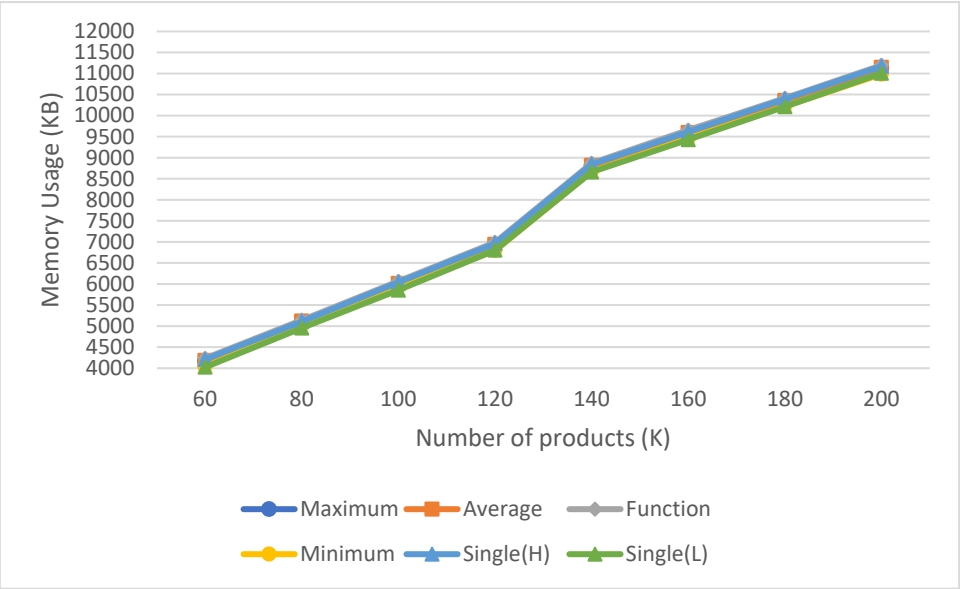

Figure 6-7: Memory usage for different dataset sizes

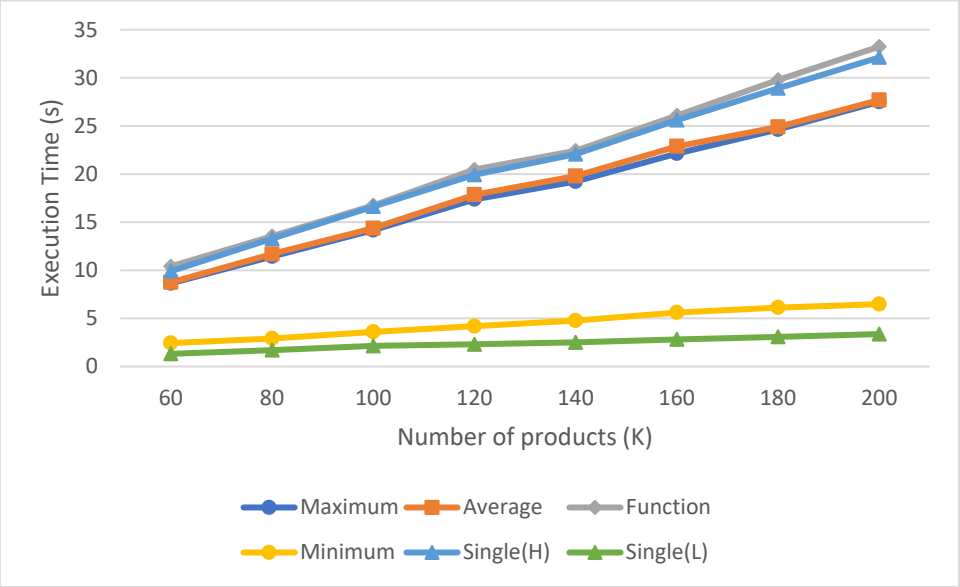

Figure 6-8: Execution time for different dataset sizes

As the number of products increases, the gain of each itemset also tends to increase. Consequently, the total profit also increases, leading to larger maximum gain thresholds for each itemset. Therefore, it cannot be guaranteed that itemsets previously classified as erasable itemsets will remain erasable itemsets as the number of products increases.

This fluctuation can increase or decrease the number of candidate and erasable itemsets, as shown in Figures 6-5 and 6-6.

In terms of memory usage, although the number of candidate and erasable itemsets does not increase significantly, increasing the number of products requires more space to store the dataset. Moreover, the disparity in the quantity of candidate and erasable itemsets among different algorithms is insufficient to impact memory usage significantly. Therefore, memory usage is primarily determined by the size of the dataset, as each algorithm operates on the same dataset, resulting in similar memory usage across all algorithms, as illustrated in Figure 6-8.

6.2.3 Performance Evaluation Using Different Numbers of Items

The experiment involving adjustments to the number of items is conducted on the P100KIxKA10 dataset. It begins with 25 items and increases sequentially by five items until reaching 60 items. The threshold interval is set to $U(0.1, 0.2)$. The experimental results are depicted in Figures 6-9 to 6-12.

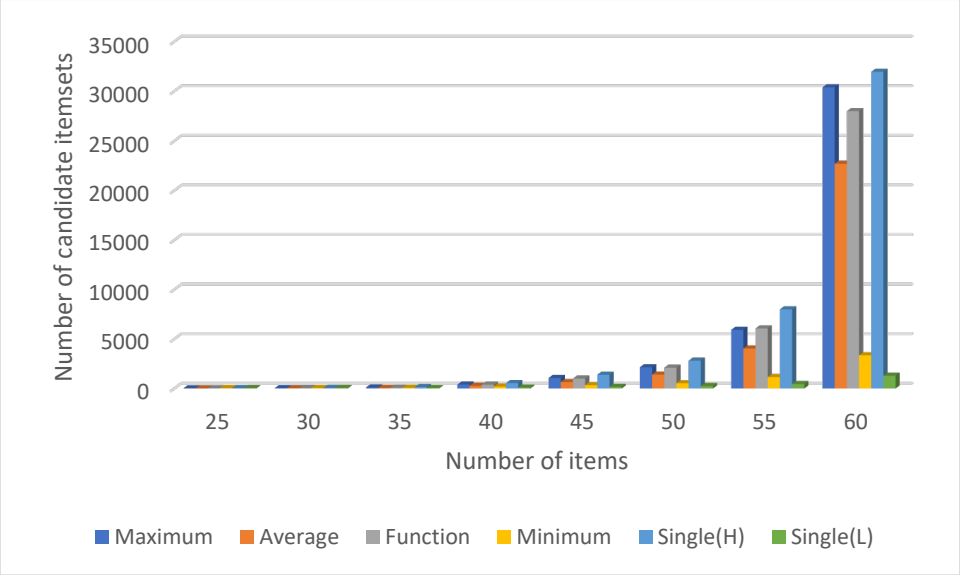

Figure 6-9: Number of candidate itemsets for different numbers of items

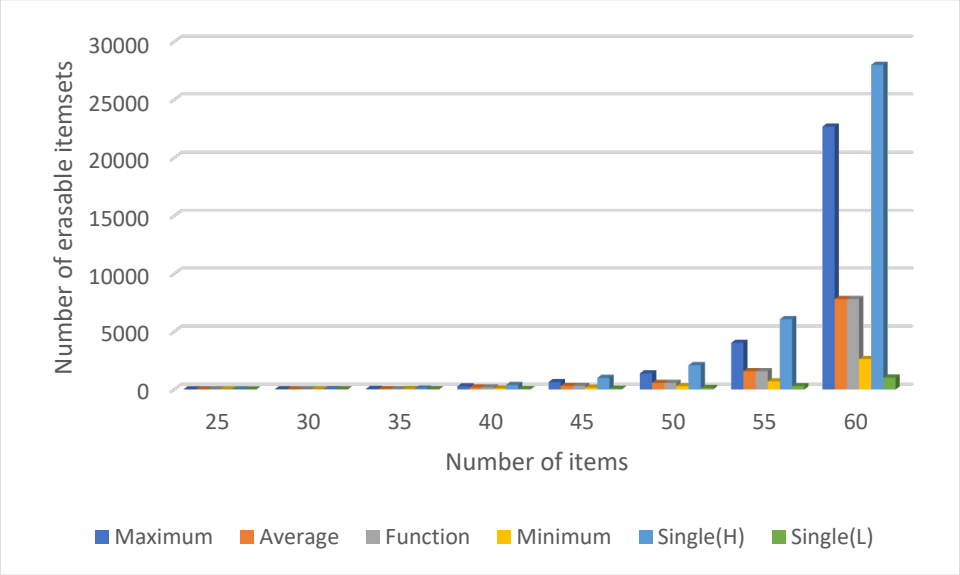

Figure 6-10: Number of erasable itemsets for different numbers of items

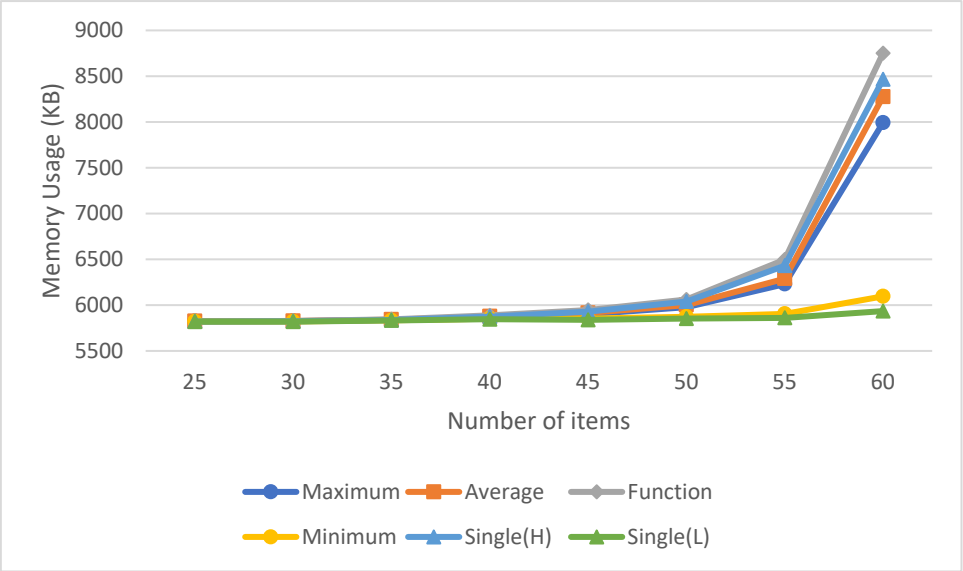

Figure 6-11: Memory usage for different numbers of items

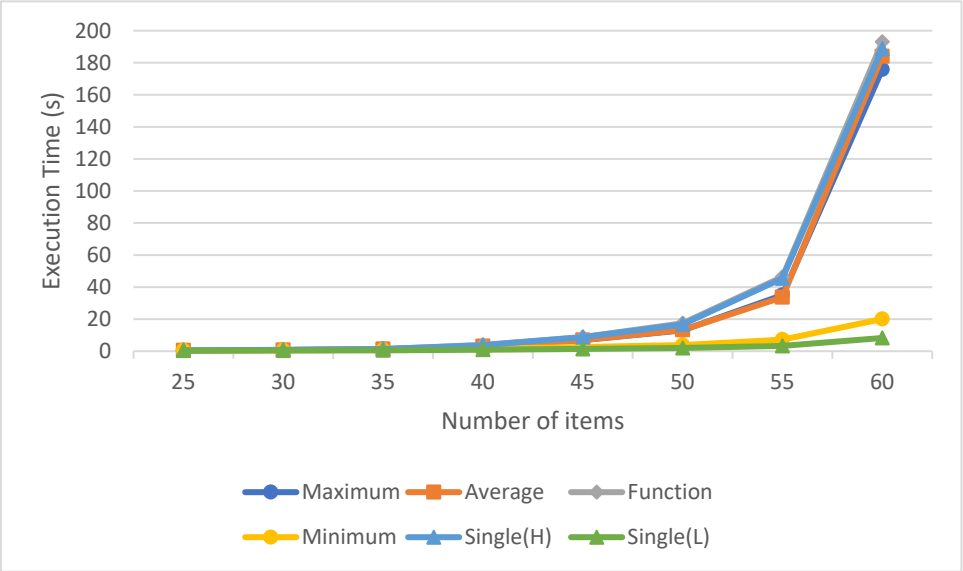

Figure 6-12: Execution time for different numbers of items

As the number of items increases, each item's probability of being utilized in a product decreases, resulting in a decrease in the gain of each itemset. Moreover, since the threshold remains constant, it becomes easier for candidate itemsets to meet the threshold and become erasable itemsets, as depicted in Figure 6-10. Additionally, with the increase in the number of items, there is a greater variety of itemset combinations

possible, leading to an exponential growth in the number of candidate itemsets, as shown in Figure 6-9.

The exponential increase in erasable and candidate itemsets leads to a greater need for storage space, resulting in increased memory usage, as depicted in Figure 6-11. Moreover, generating and validating candidate itemsets also require more time due to the increased complexity of the itemset combinations, as shown in Figure 6-12.

6.2.4 Performance Evaluation Using Different Dataset Densities

The experiment involving adjustments to the density is conducted on the dataset P100KI0.05KAx. The density starts at 25% and increases sequentially by 5% until reaching 60%. The threshold interval is set to $U(0.1, 0.2)$. The experimental results are depicted in Figures 6-13 to 6-16.

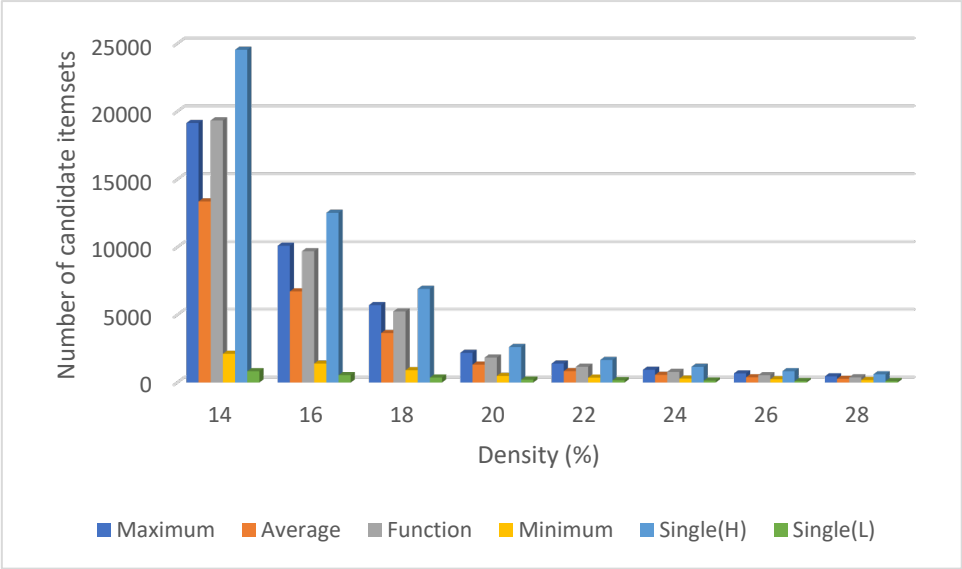

Figure 6-13: Number of candidate itemsets for different dataset densities

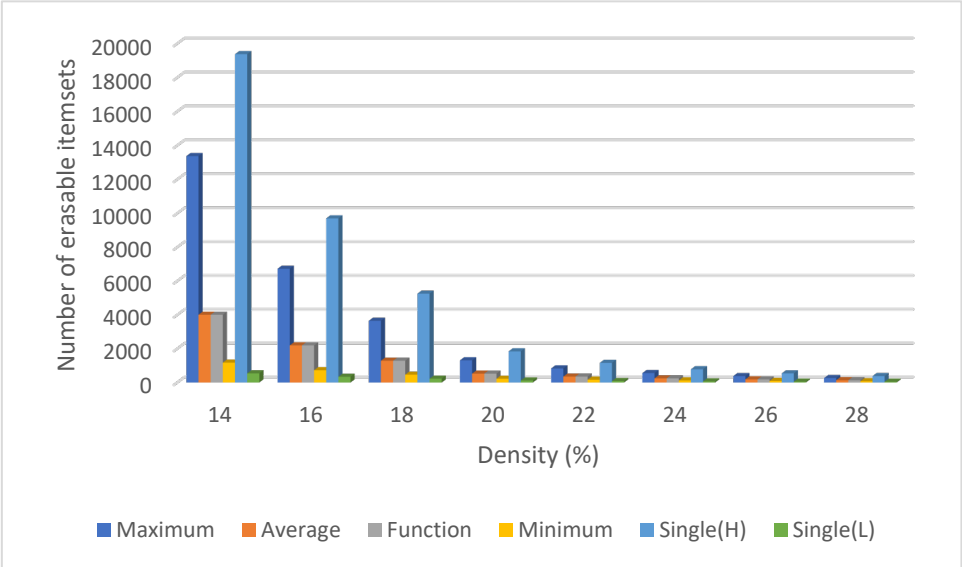

Figure 6-14: Number of erasable itemsets for different dataset densities

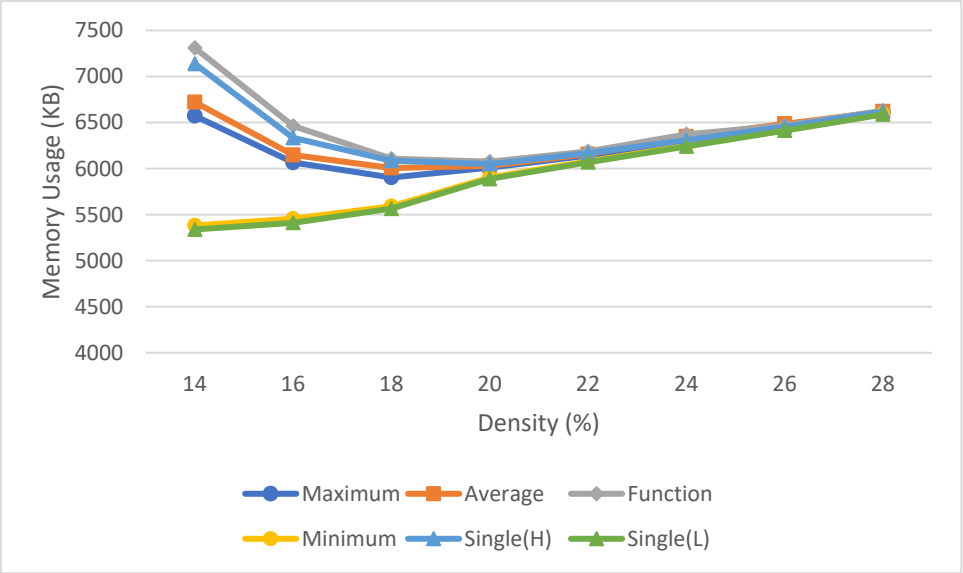

Figure 6-15: Memory usage for different dataset densities

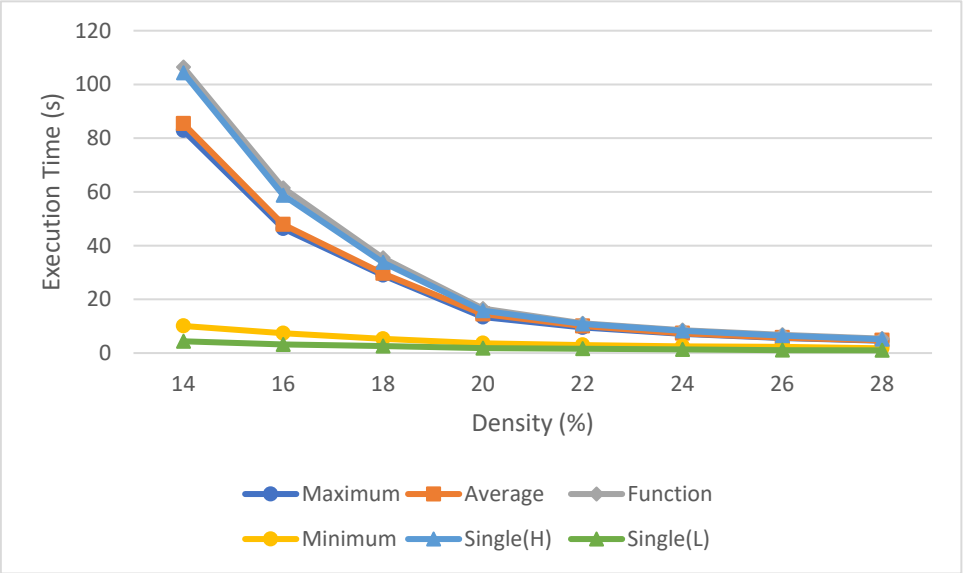

Figure 6-16: Execution time for different dataset densities

As the density increases, the probability of each item being used to produce different products rises. Consequently, the gain value of each itemset increases. However, since the threshold interval remains fixed, it becomes more difficult for itemsets to meet the threshold. This decreases the number of erasable itemsets, as shown in Figure 6-14. Moreover, reducing erasable itemsets indirectly leads to combining

fewer candidate itemsets, as depicted in Figure 6-13. The reduction in quantity also results in a significant decrease in the execution time for generating and validating candidates, as shown in Figure 6-16.

In terms of memory usage, it can be observed that the trend of the experiment is different from the previous ones, showing a V-shaped pattern. From a density of 14% to 18%, there is a downward trend in memory usage, followed by a stable increase after 20%. Memory usage depends on the dataset size, the number of candidate itemsets, and erasable itemsets. As the density increases, the dataset size increases because more items are used per product. However, in Figures 6-13 and 6-14, it can be observed that the number of candidate itemsets and erasable itemsets decreases significantly from a density of 14% to 18%. Although the dataset size increases, the sharp decline in the number of candidate and erasable itemsets results in a decrease in memory usage. This is because the increase in dataset size cannot offset the rapid decline in the number of candidate and erasable itemsets, causing memory usage to decrease. However, once the density exceeds 20%, the decrease in candidate and erasable itemsets slows down while the dataset size continues to increase steadily. At this point, memory usage shows a slower rate of increase.

6.3 Analysis of Apriori-based Versus MEI-based Methods

The experiment utilizes three real-life datasets: Mushrooms, Chess, and Connect. The objective is to compare the performance of the Apriori-based and MEI-based methods for multiple-threshold erasable itemset mining with the maximum constraint by adjusting different threshold intervals.

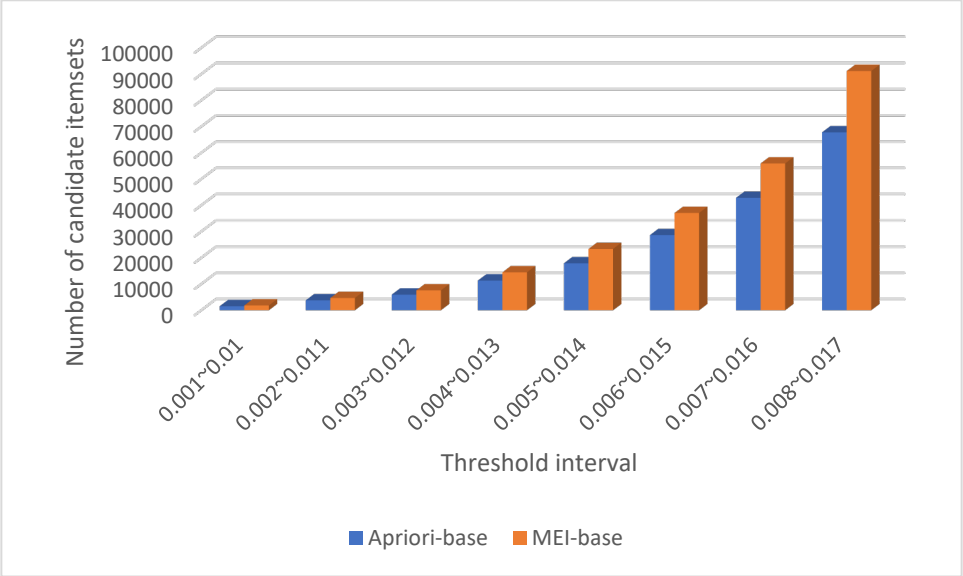

Figure 6-17: Number of candidate itemsets for the Mushrooms dataset

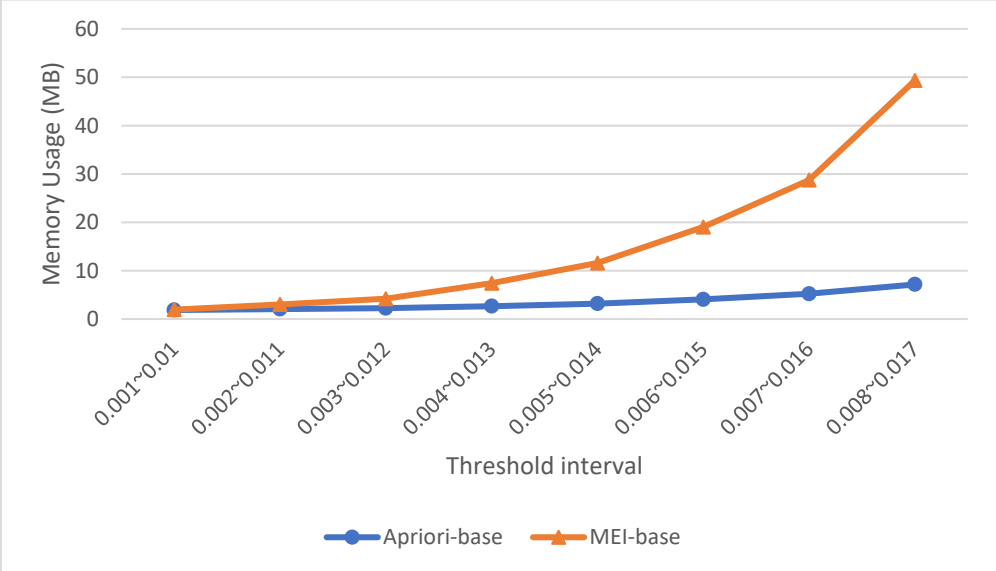

Figure 6-18: Memory usage for the Mushrooms dataset

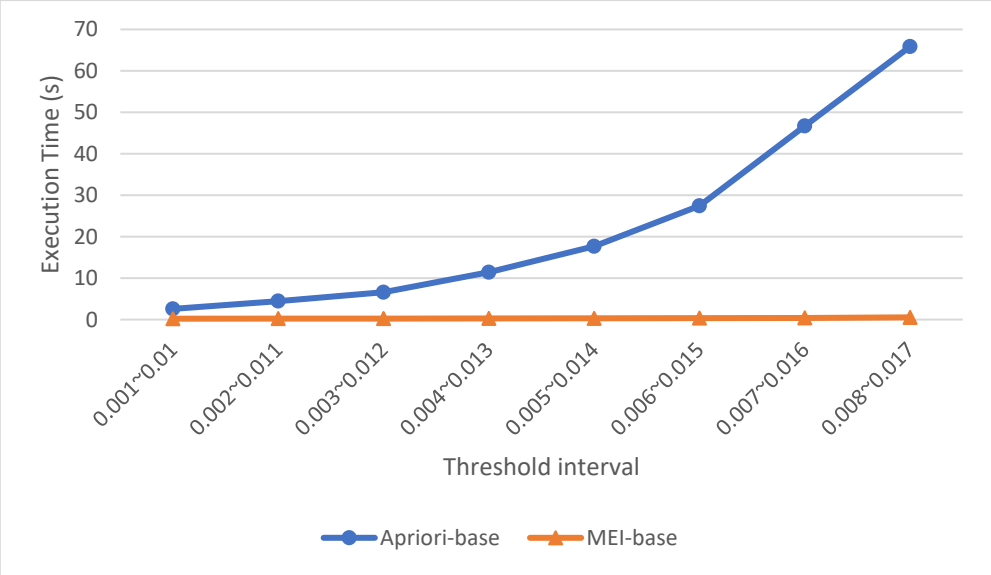

Figure 6-19: Execution time for the Mushrooms dataset

Figures 6-17 to 6-19 show the experimental results for the Mushrooms dataset. Initially, the threshold interval is set to $U(0.001, 0.01)$, with the entire interval increasing by 0.001 each time until reaching $U(0.008, 0.017)$. As the thresholds became looser, the number of candidate itemsets increased for both methods. Due to the MEI-based method not employing any pruning techniques, its number of candidate itemsets exceeded that of the Apriori-based method in Figure 6-19.

Regarding execution time, it's evident that the MEI-based method exhibits only marginal increases, and it significantly outperforms the Apriori-based method. In terms of memory usage, the MEI-based method requires additional memory due to the extra information stored for each itemset. Moreover, during tree construction, each new node creation involves a recursive call. Consequently, during each recursive call, certain

information, such as local variables, parameters, and return addresses, needs to be stored on the memory stack. As a result, the memory usage of the MEI-based method is higher compared to the Apriori-based method, as depicted in Figure 6-18.

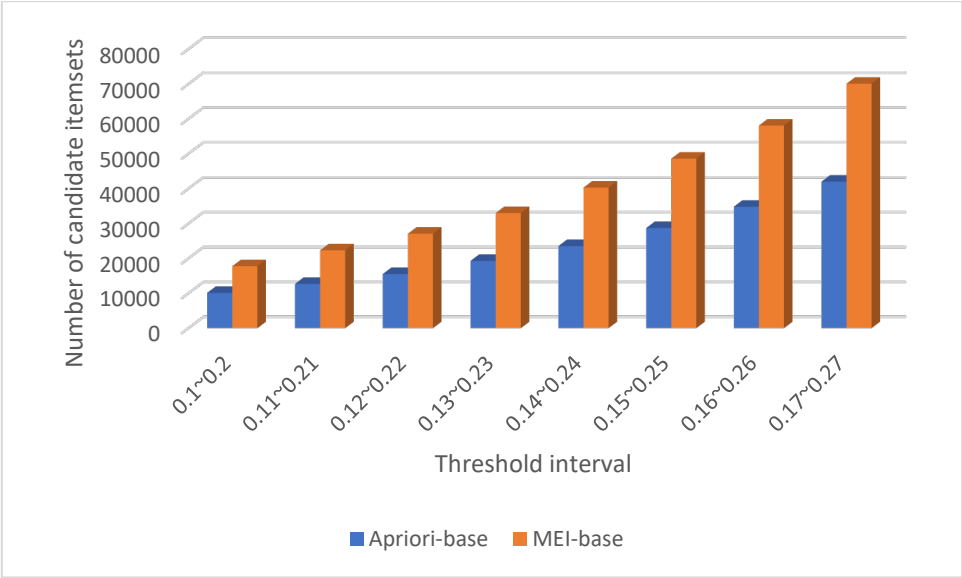

Figure 6-20: Number of candidate itemsets for the Chess dataset

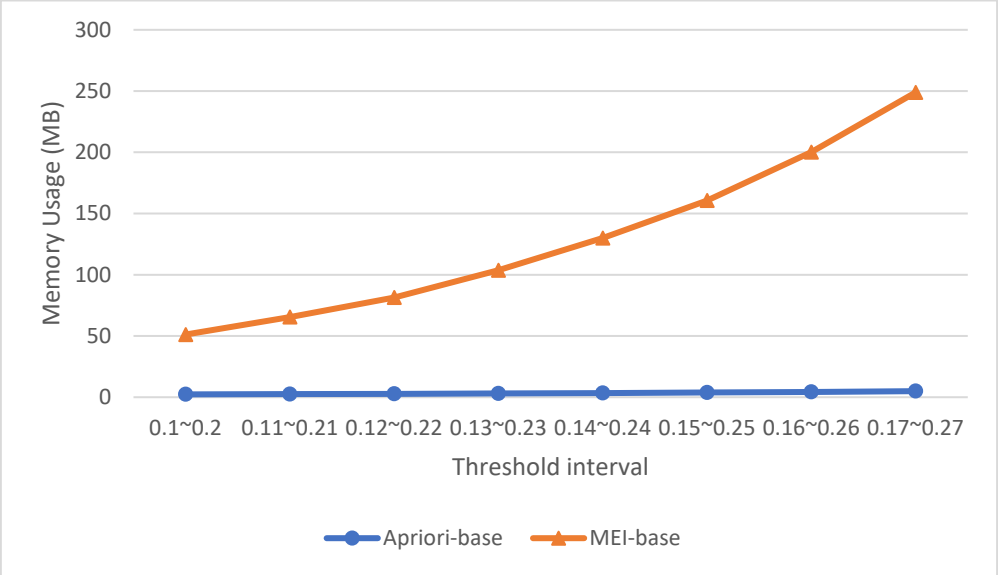

Figure 6-21: Memory usage for the Chess dataset

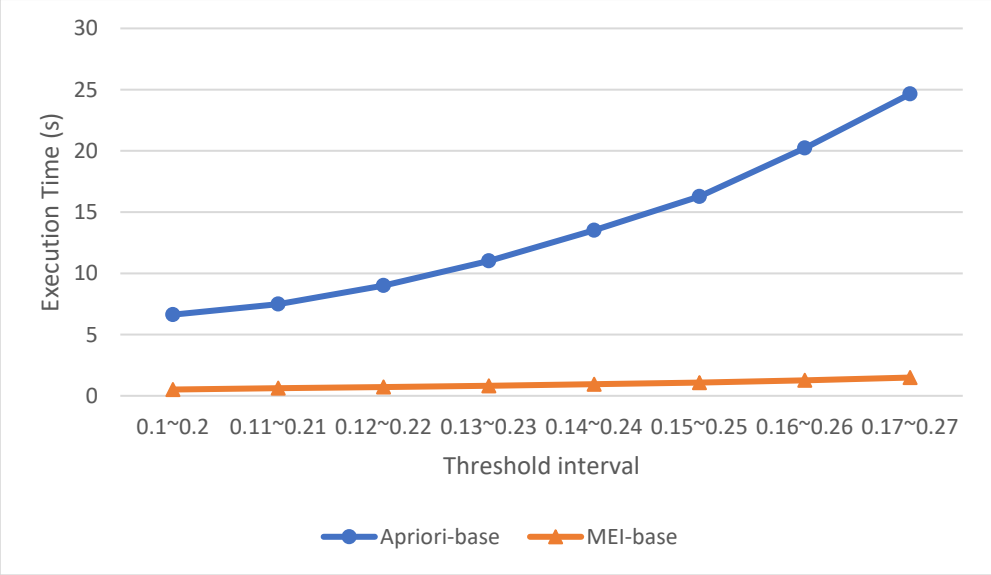

Figure 6-22: Execution time for the Chess dataset

Figures 6-20 to 6-22 show the experimental results for the Chess dataset. Initially, the threshold interval is set to $U(0.1, 0.2)$, with the entire interval increasing by 0.01 each time until reaching $U(0.017, 0.027)$. The trends are consistent with the other datasets. The memory usage of the MEI-based method is the highest among the three datasets in Figure 6-21.

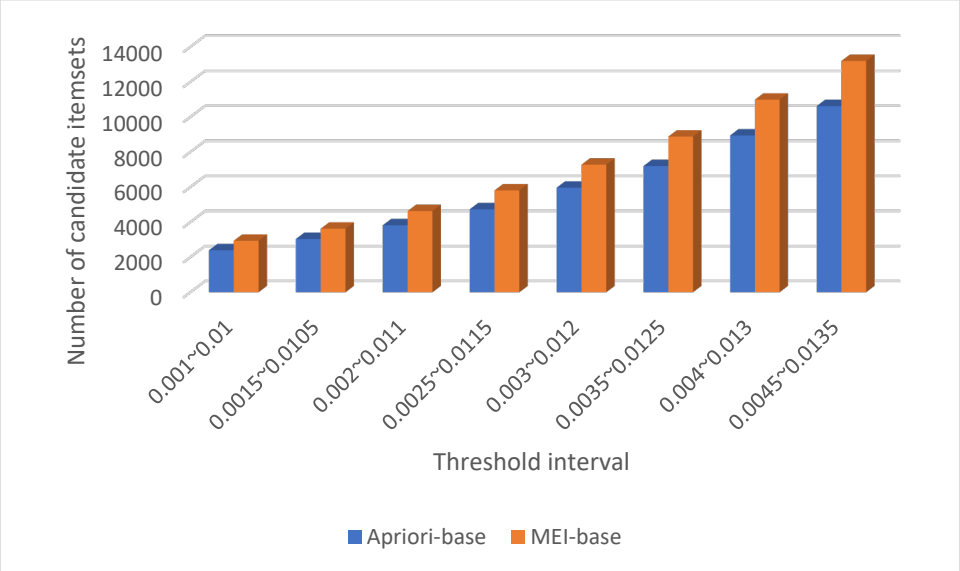

Figure 6-23: Number of candidate itemsets for the Connect dataset

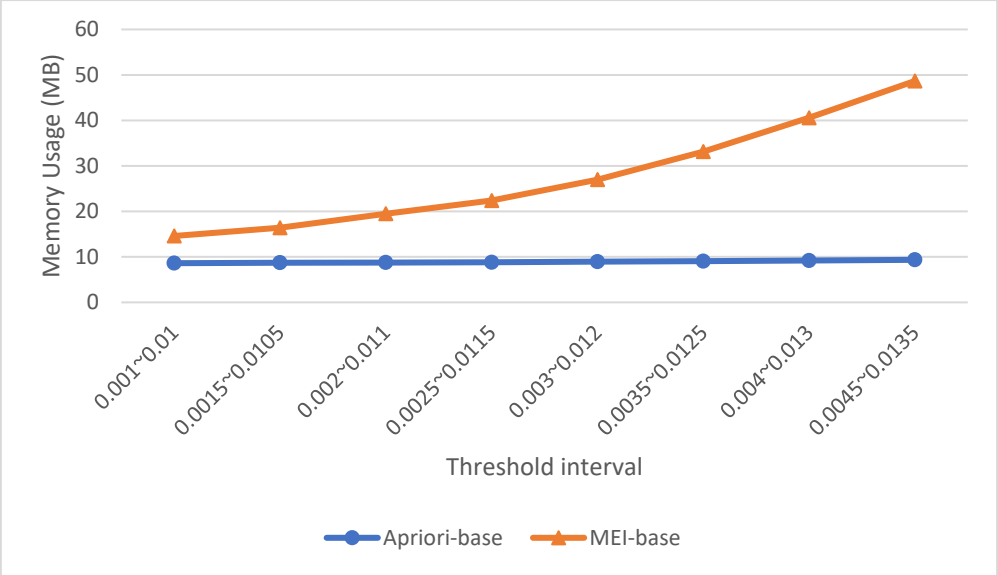

Figure 6-24: Memory usage for the Connect dataset

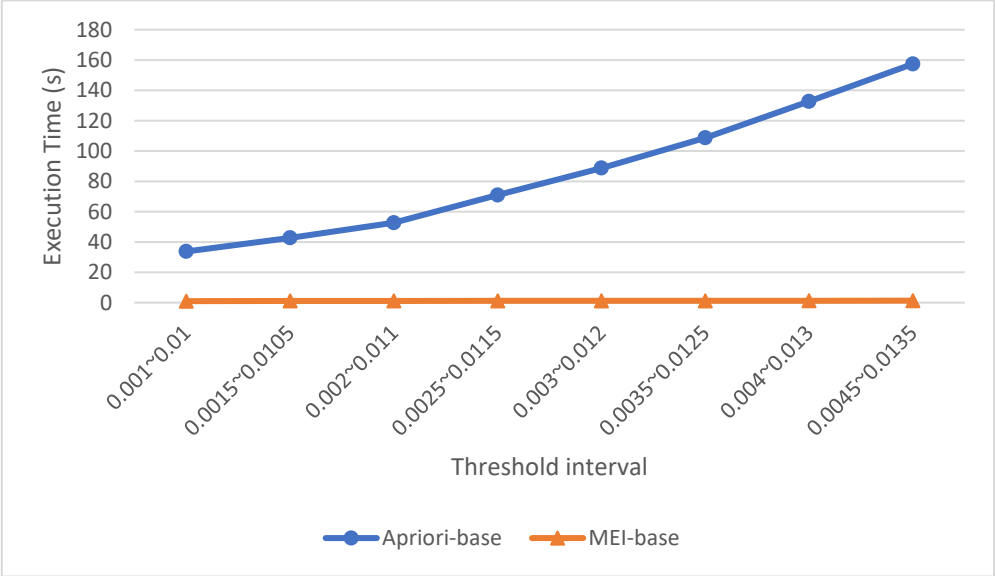

Figure 6-25: Execution time for the Connect dataset

Figures 6-23 to 6-25 show the experimental results for the Connect dataset. The Connect dataset initially set the threshold interval to $U(0.001, 0.01)$, with the entire interval increasing by 0.0005 each time until reaching $U(0.0045, 0.0135)$.

In terms of the number of candidate itemsets, the Connect dataset has the fewest among the three datasets. However, the execution time of the Apriori-based method is significantly higher compared to the other datasets, as shown in Figure 6-25. Observing the parameters of the real-life datasets in Table 6-2, it can be noted that the Connect dataset has the highest number of products. This implies that the itemsets in the Connect dataset are scanned more frequently in the dataset compared to the other datasets. From this perspective, it can be inferred that improving performance by reducing database scans is more effective than reducing the number of candidate itemsets.

Chapter 7. Conclusion and Future Work

In this thesis, we address the multiple threshold erasable itemset problem by proposing four constraint methods: minimum, average, maximum, and function. Each method uses a different approach to calculate the threshold value for each itemset. Due to the different characteristics arising from these constraints, a single algorithm cannot handle all cases. Therefore, we have designed specific algorithms for each constraint. For the minimum constraint, which has the downward closure property, we can quickly reduce the search space of candidate itemsets using methods similar to the Apriori algorithm. However, the maximum and average constraints lack the downward closure property. To handle the maximum constraint, we introduce a sorted closure and use it as the upper bound for the average constraint. We design an algorithm for the average constraint using a two-phase verification approach and extend this method to the function constraint, allowing users to flexibly set thresholds based on the current situation and environment. To address the inefficiency caused by repeatedly scanning the database, we propose a tree structure based on the MEI method, which requires only a single scan of the database, significantly improving execution speed.

In our experiments, we use two types of datasets: synthetic and real-life datasets. By adjusting one parameter at a time, we observe the impact of each algorithm under different parameters on execution time, memory usage, and the number of candidate

and erasable itemsets. Tables 7-1 and 7-2 summarize the experimental results, highlighting the differences between various constraints and methods.

Table 7-1: Comparison of experimental results under different constraints

	Minimum	**Average**	**Maximum**	**Function(Average)**
Execution time	Short	Long	Medium	Long
Memory usage	Low	High	Medium	High
Number of candidate itemsets	Low	Medium	High	High
Number of erasable itemsets	Low	Medium	High	Medium

Table 7-2: Comparison of experimental results for two mining methods

	Apriori-base	**MEI-base**
Speed	Slow	Fast
Memory usage	Low	High
Number of candidate itemsets	Low	High

Different data mining algorithms have distinct objectives. In erasable itemset mining, the primary goal is to identify combinations that minimize losses in factory management, enabling users to formulate production plans. However, different materials have varying uses and properties that need to be considered, such as cost, volume, preservation methods, and daily production capacity. Determining appropriate threshold values relies on the user's experience, as there is currently no automated mechanism for this. If the threshold is set too low, only a few materials will be mined.

Conversely, if the threshold is set too high, a large number of meaningless materials will be mined, making decision-making difficult. In the future, we will focus on exploring the relationships between each material and its influencing factors to help users set threshold values with a set of reference rules. This approach aims to provide guidelines for setting appropriate thresholds, reducing reliance on user experience and improving the effectiveness of the mining process.

References

[1] R. Agrawal, T. Imieliński, and A. Swami, "Mining association rules between sets of items in large databases," ***Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data***, pp. 207-216, 1993.

[2] R. Agrawal and R. Srikant, "Fast algorithms for mining association rules," ***Proceedings of the 20th International Conference on Very Large Data Bases, VLDB***, pp. 487-499, 1994.

[3] R. Agrawal and R. Srikant, "Mining sequential patterns," ***Proceedings of the 11th International Conference on Data Engineering***, pp. 3-14, 1995.

[4] R. Chan, Q. Yang, and Y. D. Shen, "Mining high utility itemsets," ***3rd IEEE International Conference on Data Mining***, pp. 19-19, 2003.

[5] C. H. Chen, T. P. Hong, V. S. Tseng, and C. S. Lee, "A genetic-fuzzy mining approach for items with multiple minimum supports," ***Soft Computing***, Vol. 13, pp. 521-533, 2009.

[6] C. Cooper and M. Zito, "Realistic synthetic data for testing association rule mining algorithms for market basket databases," ***11th European Conference on Principles and Practice of Knowledge Discovery in Databases (PKDD 2007)***, pp. 398-405, 2007.

[7] A. Dahbi, Y. Balouki, and T. Gadi, "Using multiple minimum support to auto-adjust the threshold of support in apriori algorithm," ***International Conference on Soft Computing and Pattern Recognition***, pp. 111-119, 2017.

[8] R. Davashi, "IME: Efficient list-based method for incremental mining of maximal erasable patterns," ***Pattern Recognition***, Vol. 148, pp. 110166, 2024.

[9] Z. Deng, "Mining top-rank-k erasable itemsets by PID_lists," ***International Journal of Intelligent Systems***, Vol. 28(4), pp. 366-379, 2013.

[10] Z. Deng and X. Xu, "An efficient algorithm for mining erasable itemsets," ***International Conference on Advanced Data Mining and Applications***, pp. 214-225, 2010.

[11] Z. H. Deng, G. D. Fang, Z. H. Wang, and X. R. Xu, "Mining erasable itemsets," ***2009 International Conference on Machine Learning and Cybernetics***, Vol. 1, pp. 67-73, 2009.

[12] Z. H. Deng and X. R. Xu, "Fast mining erasable itemsets using NC_sets," ***Expert Systems with Applications***, Vol. 39(4), pp. 4453-4463, 2012.

[13] P. Fournier-Viger, A. Gomariz, T. Gueniche, A. Soltani, C. W. Wu, and V. S. Tseng, "Spmf: a java open-source pattern mining library," ***Journal of Machine Learning Research***, Vol. 15(1), pp. 3389-3393, 2014.

[14] P. Fournier-Viger, J. C. W. Lin, A. Gomariz, T. Gueniche, A. Soltani, Z. Deng, and H. T. Lam, "The SPMF open-source data mining library version 2," ***Machine Learning and Knowledge Discovery in Databases: European Conference (ECML PKDD 2016)***, pp. 36-40, 2016.

[15] G. Gupta and H. Aggarwal, "Improving customer relationship management using data mining," ***International Journal of Machine Learning and Computing***, Vol. 2(6), pp. 874-877, 2012.

[16] J. Han, J. Pei, and Y. Yin, "Mining frequent patterns without candidate generation," ***ACM Sigmod Record***, Vol. 29(2), pp. 1-12, 2000.

[17] T. P. Hong, H. Chang, S. M. Li, and Y. C. Tsai, "A unified temporal erasable itemset mining approach," ***2021 International Conference on Technologies and Applications of Artificial Intelligence (TAAI)***, pp. 194-198, 2021.

[18] T. P. Hong, H. Chang, S. M. Li, and Y. C. Tsai, "A dedicated temporal erasable-itemset mining algorithm," ***International Conference on Intelligent Systems Design and Applications***, pp. 977-985, 2022.

[19] T. P. Hong, Y. C. Chang, W. M. Huang, and W. Y. Lin, "Multiple-threshold erasable mining under the tightest constraint," ***International Conference on Intelligent Information Hiding and Multimedia Signal Processing***, pp. 369-377, 2022.

[20] T. P. Hong, Y. C. Chang, C. H. Wang, and W. M. Huang, "Mining erasable itemsets with multiple thresholds under the loose constraint," ***The 9th International Conference on Advances in Computation, Communications and Services***, pp. 9-14, 2024.

[21] T. P. Hong, H. W. Chen, W. M. Huang, and C. H. Chen, "Erasable pattern mining with quantitative information," ***2019 International Conference on Technologies and Applications of Artificial Intelligence (TAAI)***, pp. 1-4, 2019.

[22] T. P. Hong, L. H. Chen, S. L. Wang, C. W. Lin, and B. Vo, "Quasi-erasable itemset mining," ***2017 IEEE International Conference on Big Data***, pp. 1816-1820, 2017.

[23] T. P. Hong, Y. L. Chen, W. M. Huang, and Y. C. Tsai, "Erasable-itemset mining for sequential product databases," ***International Conference on Hybrid Intelligent Systems***, pp. 566-574, 2022.

[24] T. P. Hong, C. C. Chiu, J. H. Su, and C. H. Chen, "Applicable metamorphic testing for erasable-itemset mining," ***IEEE Access***, Vol. 10, pp. 38545-38554, 2022.

[25] T. P. Hong, W. M. Huang, G. C. Lan, M. C. Chiang, and J. C. W. Lin, "A bitmap approach for mining erasable itemsets," ***IEEE Access***, Vol. 9, pp. 106029-106038, 2021.

[26] T. P. Hong, C. C. Li, S. L. Wang, and C. W. Lin, "Maintenance of erasable itemsets for product deletion," ***Proceedings of the 5th Multidisciplinary International Social Networks Conference***, pp. 1-4, 2018.

[27] T. P. Hong, C. C. Li, S. L. Wang, and J. C. W. Lin, "Reducing database scan in maintaining erasable itemsets from product deletion," ***2018 IEEE International Conference on Big Data***, pp. 2627-2632, 2018.

[28] T. P. Hong, J. X. Li, Y. C. Tsai, and W. M. Huang, "Unified temporal erasable itemset mining with a lower-bound strategy," ***2022 IEEE International Conference on Big Data***, pp. 6207-6211, 2022.

[29] T. P. Hong, J. X. Li, Y. C. Tsai, and W. M. Huang, "Tree-based unified temporal erasable-itemset mining," ***Asian Conference on Intelligent Information and Database Systems***, pp. 224-233, 2023.

[30] T. P. Hong, K. Y. Lin, C. W. Lin, and B. Vo, "An incremental mining algorithm for erasable itemsets," ***2017 IEEE International Conference on INnovations in Intelligent SysTems and Applications (INISTA)***, pp. 286-289, 2017.

[31] T. P. Hong, T. T. Shih, C. W. Lin, and B. Vo, "Merging two sets of erasable itemsets," ***2016 International Computer Symposium (ICS)***, pp. 90-93, 2016.

[32] T. P. Hong, C. H. Wang, C. C. Li, and W. Y. Lin, "Reduction of erasable itemset mining to frequent itemset mining," ***Proceedings of the Annual Conference of JSAI 33rd***, pp. 2K3E102-2K3E102, 2019.

[33] T. P. Hong, C. H. Wang, W. Y. Lin, and S. L. Wang, "Reduction of frequent-itemset mining to erasable-itemset mining," ***The 2019 International Conference on Technologies and Applications of Artificial Intelligence (TAAI)***, 2019.

[34] H. W. Hu, H. C. Chang, and W. S. Lin, "An optimized frequent pattern mining algorithm with multiple minimum supports," ***2016 IEEE International Conference on Big Data***, pp. 1734-1741, 2016.

[35] Y. H. Hu and Y. L. Chen, "Mining association rules with multiple minimum supports: a new mining algorithm and a support tuning mechanism," ***Decision Support Systems***, Vol. 42(1), pp. 1-24, 2006.

[36] T. C. K. Huang, "Discovery of fuzzy quantitative sequential patterns with multiple minimum supports and adjustable membership functions," ***Information Sciences***, Vol. 222, pp. 126-146, 2013.

[37] W. M. Huang, T. P. Hong, M. C. Chiang, and J. C. W. Lin, "Using multi-conditional minimum thresholds in temporal fuzzy utility mining,"

International Journal of Computational Intelligence Systems, Vol. 12(2), pp. 613-626, 2019.

[38] W. M. Huang, T. P. Hong, G. C. Lan, M. C. Chiang, and J. C. W. Lin, "Temporal-based fuzzy utility mining," ***IEEE Access***, Vol. 5, pp. 26639-26652, 2017.

[39] W. M. Huang, T. P. Hong, J. C. W. Lin, and C. H. Chen, "Using Multiple Thresholds on Temporal Fuzzy Utility Mining with Maximum Constraint," ***TANET 2019***, pp. 345-349, 2019.

[40] M. E. Kara, S. ü. O. Fırat, and A. Ghadge, "A data mining-based framework for supply chain risk management," ***Computers and Industrial Engineering***, Vol. 139, pp. 105570, 2020.

[41] R. U. Kiran and P. K. Reddy, "Novel techniques to reduce search space in multiple minimum supports-based frequent pattern mining algorithms," ***Proceedings of the 14th International Conference on Extending Database Technology***, pp. 11-20, 2011.

[42] S. Krishnamoorthy, "Efficient mining of high utility itemsets with multiple minimum utility thresholds," ***Engineering Applications of Artificial Intelligence***, Vol. 69, pp. 112-126, 2018.

[43] G. C. Lan, T. P. Hong, Y. H. Lin, and S. L. Wang, "Fast discovery of high fuzzy utility itemsets," ***2014 IEEE International Conference on Systems, Man, and Cybernetics (SMC)***, pp. 2764-2767, 2014.

[44] T. Le and B. Vo, "MEI: an efficient algorithm for mining erasable itemsets," ***Engineering Applications of Artificial Intelligence***, Vol. 27, pp. 155-166, 2014.

[45] T. Le, B. Vo, and F. Coenen, "An efficient algorithm for mining erasable itemsets using the difference of NC-Sets," ***2013 IEEE International Conference on Systems, Man, and Cybernetics***, pp. 2270-2274, 2013.

[46] T. Le, B. Vo, and G. Nguyen, "A survey of erasable itemset mining algorithms," ***Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery***, Vol. 4(5), pp. 356-379, 2014.

[47] G. Lee, U. Yun, and H. Ryang, "Mining weighted erasable patterns by using underestimated constraint-based pruning technique," ***Journal of Intelligent and Fuzzy Systems***, Vol. 28(3), pp. 1145-1157, 2015.

[48] G. Lee, U. Yun, H. Ryang, and D. Kim, "Erasable itemset mining over incremental databases with weight conditions," ***Engineering Applications of Artificial Intelligence***, Vol. 52, pp. 213-234, 2016.

[49] Y. C. Lee, T. P. Hong, and W. Y. Lin, "Mining fuzzy association rules with multiple minimum supports using maximum constraints," ***Knowledge-Based***

Intelligent Information and Engineering Systems: 8th International Conference (KES 2004), pp. 1283-1290, 2004.

[50] Y. C. Lee, T. P. Hong, and W. Y. Lin, "Mining association rules with multiple minimum supports using maximum constraints," ***International Journal of Approximate Reasoning***, Vol. 40(1-2), pp. 44-54, 2005.

[51] Y. C. Lee, T. P. Hong, and T. C. Wang, "Mining fuzzy multiple-level association rules under multiple minimum supports," ***2006 IEEE International Conference on Systems, Man and Cybernetics***, Vol. 5, pp. 4112-4117, 2006.

[52] J. C. W. Lin, W. Gan, P. Fournier-Viger, and T. P. Hong, "Mining high-utility itemsets with multiple minimum utility thresholds," ***Proceedings of the 8th International Conference on Computer Science and Software Engineering***, pp. 9-17, 2015.

[53] J. C. W. Lin, W. Gan, P. Fournier-Viger, T. P. Hong, and J. Zhan, "Efficient mining of high-utility itemsets using multiple minimum utility thresholds," ***Knowledge-Based Systems***, Vol. 113, pp. 100-115, 2016.

[54] B. Liu, W. Hsu, and Y. Ma, "Mining association rules with multiple minimum supports," ***Proceedings of the 5th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining***, pp. 337-341, 1999.

[55] Y.-C. Liu, C. P. Cheng, and V. S. Tseng, "Discovering relational-based association rules with multiple minimum supports on microarray datasets," ***Bioinformatics***, Vol. 27(22), pp. 3142-3148, 2011.

[56] Y. Liu, W. K. Liao, and A. Choudhary, "A fast high utility itemsets mining algorithm," ***Proceedings of the 1st International Workshop on Utility-based Data Mining***, pp. 90-99, 2005.

[57] Y. Liu, W. K. Liao, and A. Choudhary, "A two-phase algorithm for fast discovery of high utility itemsets," ***Pacific-Asia Conference on Knowledge Discovery and Data Mining***, pp. 689-695, 2005.

[58] H. Mannila, "Database methods for data mining," ***Proceedings of the 4th International Conference on Knowledge Discovery and Data Mining (KDD-98) Tutorial***, 1998.

[59] H. Nam, U. Yun, E. Yoon, and J. C. W. Lin, "Efficient approach for incremental weighted erasable pattern mining with list structure," ***Expert Systems with Applications***, Vol. 143, pp. 113087, 2020.

[60] R. T. Ng, L. V. Lakshmanan, J. Han, and A. Pang, "Exploratory mining and pruning optimizations of constrained associations rules," ***ACM Sigmod Record***, Vol. 27(2), pp. 13-24, 1998.

[61] E. W. Ngai, L. Xiu, and D. C. Chau, "Application of data mining techniques in customer relationship management: a literature review and classification," ***Expert Systems with Applications***, Vol. 36(2), pp. 2592-2602, 2009.

[62] G. Nguyen, T. Le, B. Vo, and B. Le, "A new approach for mining top-rank-k erasable itemsets," ***Asian Conference on Intelligent Information and Database Systems***, pp. 73-82, 2014.

[63] L. Nguyen, G. Nguyen, and B. Le, "Fast algorithms for mining maximal erasable patterns," ***Expert Systems with Applications***, Vol. 124, pp. 50-66, 2019.

[64] J. Pei, J. Han, and L. V. Lakshmanan, "Mining frequent itemsets with convertible constraints," ***Proceedings 17th International Conference on Data Engineering***, pp. 433-442, 2001.

[65] J. Pei, J. Han, and L. V. Lakshmanan, "Pushing convertible constraints in frequent itemset mining," ***Data Mining and Knowledge Discovery***, Vol. 8(3), pp. 227-252, 2004.

[66] Y. Pei and O. Zaïane, "A synthetic data generator for clustering and outlier analysis," pp. 1-33, 2006.

[67] D. D. Raj and M. Ranganathan, "A comprehensive survey on erasable itemset mining," ***International Journal of Computer Science and Information Security (IJCSIS)***, Vol. 15(7), pp. 184-201, 2017.

[68] Z. B. Ren and Q. Zhang, "Pushing fuzzy constraints in frequent itemset mining," ***2006 International Conference on Machine Learning and Cybernetics***, pp. 1373-1377, 2006.

[69] C. Rygielski, J. C. Wang, and D. C. Yen, "Data mining techniques for customer relationship management," ***Technology in Society***, Vol. 24(4), pp. 483-502, 2002.

[70] T. Saeed, "The application of data mining techniques for financial risk management: a classification framework," ***International Journal of Computer Science and Network Security (IJCSNS)***, Vol. 20(8), pp. 84, 2020.

[71] K. K. Sethi and D. Ramesh, "High average-utility itemset mining with multiple minimum utility threshold: a generalized approach," ***Engineering Applications of Artificial Intelligence***, Vol. 96, pp. 103933, 2020.

[72] M. C. Tseng and W. Y. Lin, "Efficient mining of generalized association rules with non-uniform minimum support," ***Data and Knowledge Engineering***, Vol. 62(1), pp. 41-64, 2007.

[73] S. Tsumoto and S. Hirano, "Risk mining in medicine: Application of data mining to medical risk management," ***Fundamenta Informaticae***, Vol. 98(1), pp. 107-121, 2010.

[74] B. Vo, T. Le, G. Nguyen, and T. P. Hong, "Efficient algorithms for mining erasable closed patterns from product datasets," ***IEEE Access***, Vol. 5, pp. 3111-3120, 2017.

[75] B. Vo, T. Le, W. Pedrycz, G. Nguyen, and S. W. Baik, "Mining erasable itemsets with subset and superset itemset constraints," ***Expert Systems with Applications***, Vol. 69, pp. 50-61, 2017.

[76] S. Wan, W. Gan, X. Guo, J. Chen, and U. Yun, "FUIM: fuzzy utility itemset mining," ***arXiv:2111.00307***, 2021.

[77] K. Wang, Y. He, and J. Han, "Pushing support constraints into association rules mining," ***IEEE Transactions on Knowledge and Data Engineering***, Vol. 15(3), pp. 642-658, 2003.

[78] J. Wu, L. Zhang, W. Cui, and B. Jiang, "Frequent pattern mining algorithm based on multi minimum support," ***2019 IEEE International Conference on Power Data Science (ICPDS)***, pp. 130-134, 2019.

[79] K. J. Yang, G. C. Lan, T. P. Hong, and Y. M. Chen, "Partial periodic patterns mining with multiple minimum supports," ***2013 9th International Conference on Information, Communications and Signal Processing***, pp. 1-4, 2013.