TRƯỜNG ĐẠI HỌC FPT

Capstone Project Report

Mining Correlated High-Utility Itemsets Using the Cosine Measure

Students	**1. Huynh Anh Duy (HE153764)** **2. Huynh Anh Khoa (HE153759)**
Supervisor	**Associate Professor Phan Duy Hung**

Bachelor of Artificial Intelligence
Fpt University – Hoa Lac Campus

Summer 2023

Acknowledgement

This project is the result of our dedicated work over an extended period. We would like to express our sincerest gratitude to Associate Professor Phan Duy Hung, who has enthusiastically guided us and provided us with opportunities to engage in scientific research. This project would not have been completed without the valuable knowledge and experiences gained from these wonderful opportunities. We thank Dr. Tran Van Ha, the lecturer of Data Mining, for his dedicated assistance whenever we had questions related to the subject. We also extend our thanks to FPT University for providing us with excellent learning resources and a high-quality academic environment. Additionally, we are deeply grateful to our families and loved ones for their unwavering support. Despite our utmost efforts, we acknowledge that this project may still have some shortcomings. Our team is more than willing to embrace any feedback and suggestions to continuously improve ourselves.
Thank you very much!

Abstract

High utility itemset mining (HUIM) is a problem posed to find itemsets in transaction database with high utility. However, using only utility as selection criterion makes most of the found itemsets have a very low correlation between their items, therefore it cannot be effectively applied in practice. Fast correlation high-utility itemset miner (FCHM) is an efficiency algorithm that applies correlation to HUIM problem to discover correlated high-utility itemsets (CHIs). The correlation measures used in FCHM include bond and all-confidence. This thesis proposes a new version of FCHM algorithm by using cosine measure to calculate correlation between items which is FCHM_{cosine}. Experimental results on three benchmark real-life datasets show that the proposed algorithm not only significantly reduces weakly correlated itemsets but also improves running time and memory consumption.

Keywords: high-utility itemset mining, correlated high-utility itemsets, correlation, cosine measure

Table content

List of Figures

List of Tables

Chapter 1. Introduction

1.1 Basic concepts

The primary objective of data mining is to extract patterns or develop models from databases to comprehend historical trends or make future predictions. Various data mining algorithms have been proposed for data analysis [1]. Some algorithms generate models that function as black boxes, such as certain types of neural networks that exhibit high predictive accuracy but lack interpretability for humans. To extract knowledge from data that can be understood by humans, pattern mining algorithms are designed. The goal is to discover patterns in data that are interesting, useful, and/or unexpected. One advantage of pattern mining over several other data mining approaches is that discovering patterns is a type of unsupervised learning as it does not require labeled data. Patterns can be directly extracted from raw data and used to understand data and support decision-making. Pattern mining algorithms have been designed to extract various types of patterns, each providing different information to the user, and for extracting patterns from different types of data. Popular types of patterns include sequential patterns [3], itemsets [4], clusters, trends, outliers, and graph structures [2].

Research on pattern mining algorithms started in the 1990s with algorithms aimed at discovering frequent patterns in databases. The first algorithm for frequent pattern mining is Apriori [5]. It is designed to discover frequent itemsets in customer transaction databases. A transaction database is a set of records (transactions) indicating the items purchased by customers at different times. A frequent itemset refers to a group of values (items) that is frequently purchased by customers and appears in many transactions within a transaction database. For example, a frequent itemset in a database may be that many customers buy the items "noodles" and "spicy sauce." Such patterns are easily understandable by humans and can be used to support decision-making. For instance, the pattern {noodles, spicy sauce} can be utilized to make marketing decisions, such as co-promoting noodles with spicy sauce. The discovery of frequent itemsets is a well-studied data mining task and has applications in numerous domains. It can be viewed as the general task of analyzing a database to find co-occurring values (items) in a set of database records (transactions) [6–13].

Although frequent pattern mining is useful, it relies on the assumption that frequent patterns are inherently interesting. However, this assumption does not hold for numerous applications. For example, in a transaction database, the pattern {milk, bread} may be highly frequent but uninteresting as it represents a common purchase behavior that may yield low profitability. On the other hand, several patterns such as {caviar, champagne} may not be frequent but can yield higher profits. Hence, to identify interesting patterns in data, other aspects such as profit or utility can be considered.

To overcome the limitations of frequent itemset mining, there has been a growing research focus on the exploration of high utility patterns in databases [14–21]. The objective of utility mining is to identify patterns that possess significant utility or importance to the user, where the utility of a pattern is expressed through a utility function. This function can be defined based on various criteria such as the profitability generated by an item's sale or the amount of time spent on webpages. Numerous types of high utility patterns have been investigated, with a particular emphasis on high utility itemsets [19]. Mining high utility itemsets can be viewed as an extension of the frequent itemset mining problem. In this case, the input is a transaction database where each item is assigned a weight representing its importance, and items can have non-binary quantities in transactions. This generalized problem formulation enables the modeling of various tasks, such as discovering itemsets that yield substantial profits in transaction databases, identifying sets of webpages where users spend a significant amount of time, or finding all frequent patterns as in traditional frequent pattern mining. In general, high utility itemset mining is currently a highly active research area.

1.2 Problem Definition

In this section, we begin by introducing the concept of frequent itemset mining, followed by an exploration of its generalization known as high utility itemset mining. We subsequently delve into the key properties of the problem of high utility itemset mining, highlighting the distinctions it holds in comparison to frequent itemset mining.

1.2.1 Frequent Itemset Mining

The task of frequent itemset mining involves extracting patterns from a transaction database, where each transaction is a collection of items or symbols. More formally, a transaction database D is defined as a set of records (transactions), represented as $D = \{T_1, T_2, \dots, T_n\}$, where each transaction T_q is a subset of items $T_q \subseteq I$ and is uniquely identified by its TID (Transaction IDentifier) q. For example, consider Table 1.1, which represents a customer transaction database with five transactions labeled as T_1, T_2, T_3, T_4, T_5. In Table 1.3, transaction T_3 indicates that a customer made a purchase involving items x_1, x_3 and x_4.

Table 1.1. An example of transaction database

TID	Transaction
T_1	$\{x_1, x_2, x_3, x_4, x_5\}$
T_2	$\{x_2, x_3, x_4, x_5\}$
T_3	$\{x_1, x_3, x_4\}$
T_4	$\{x_1, x_3, x_5\}$
T_5	$\{x_2, x_3, x_5\}$

Table 1.2. List frequent itemsets satisfy *minsup* = 3

Itemset	Support
$\{x_1$	3
$\{x_2\}$	3
$\{x_3\}$	5
$\{x_4\}$	3
$\{x_5\}$	4
$\{x_1, x_3\}$	3
$\{x_2, x_3\}$	3
$\{x_3, x_4\}$	3
$\{x_2, x_5\}$	3
$\{x_3, x_5\}$	4
$\{x_2, x_3, x_5\}$	3

Table 1.3. An example of quantitative transaction database

TID	Transaction
T_1	$(x_1, 1), (x_2, 5), (x_3, 1), (x_4, 3), (x_5, 1)$
T_2	$(x_2, 4), (x_3, 3), (x_4, 3), (x_5, 1)$
T_3	$(x_1, 1), (x_3, 1), (x_4, 1)$
T_4	$(x_1, 2), (x_3, 6), (x_5, 2)$
T_5	$(x_2, 2), (x_3, 2), (x_5, 1)$

The primary objective of frequent itemset mining is to identify itemsets (sets of items) that exhibit high support, meaning they appear frequently. Mathematically, an *itemset X* is a finite collection of items, denoted as $X \subseteq I$. The notation $|X|$ represents the cardinality of the itemset, indicating the number of items it contains. Specifically, an itemset *X* is considered a *k*-itemset if it comprises *k* items $|X| = k$. For instance, $\{x_1, x_2, x_3\}$ is a 3-itemset, while $\{x_1, x_2\}$ is a 2-itemset. The support measure is defined as follows.

Definition 1 *(Support value)*: The support (frequency) of an itemset *X* in a transaction database *D* is represented as *sup(X)* and is defined as $sup(X) = |\{T | X \subseteq T \wedge T \in D\}|$, which means it is the count of transactions that contain the itemset *X*. For instance, in the given Table 1.3 database, the support of the itemset $\{x_1, x_3\}$ is 3 because it appears in three transactions (T_1, T_3 and T_4). This definition of support value is known as relative support. Another equivalent definition is to express the support as a percentage of the total number of transactions, which is referred to as *absolute support*. In the case of $\{x_1, x_3\}$, 3 in total 5 transactions has this itemset, so the absolute support is 60%. The problem of frequent itemset mining is defined as follows:

Definition 2 *(Frequent itemset)*: Considering a user-defined threshold *minsup* > 0, an itemset X is classified as a *frequent itemset* if its support *sup(X)* is equal to or exceeds the *minsup* threshold (i.e., $sup(X) \geq minsup$). If the support is below the *minsup* threshold, *X* is categorized as an *infrequent itemset*.

Definition 3 *(Problem definition):* The objective of *frequent itemset mining* is to identify all itemsets that meet the frequency criterion in a transaction database *D*, using the *minsup* threshold specified by the user.

As an example, considering Table 1.3 database with a *minsup* of 3, there are a total of 11 itemsets that qualify as frequent, as shown in Table 1.2.

Over the past few decades, researchers have extensively studied the problem of frequent itemset mining. Several algorithms have been suggested for effectively identifying frequent patterns, such as Apriori [5], FP-Growth [22], Eclat [23], LCM [24], and H-Mine [25]. While frequent itemset mining finds applications in various domains, it assumes that frequent patterns are inherently useful or interesting to the user, which is not always the case. To overcome this constraint, traditional frequent pattern mining has been expanded to high utility itemset mining. This approach involves assigning numerical values to items and selecting patterns based on a utility function defined by the user.

1.2.2 High Utility Itemset Mining

The task of high utility itemset mining involves discovering patterns in a specialized type of transaction database known as a quantitative transaction database. This type of database provides additional information, such as the quantities of items in transactions and weights indicating the relative importance of each item to the user.

In precise terms, a quantitative transaction database *D* is defined as follows. Consider the set *I*, which represents all items, expressed as $I = \{i_1, i_2, \dots, i_m\}$. A quantitative transaction database D comprises a collection of transactions denoted as $D = \{T_0, T_1, \dots, T_n\}$. Each transaction T_q is a set of items (i.e., $T_q \subseteq I$) and possesses a unique identifier q known as its TID (Transaction IDentifier). Every item i ∈ I is linked to a positive number *p(i)*, referred to as its *external utility*, indicating its relative significance for the user. Moreover, for each item *k* that appears in a transaction T_d, there exists a positive number $q(k, T_d)$ known as its *internal utility*, representing the quantity of item *k* in transaction T_d.

To illustrate these definitions, let's consider a practical example of a customer transaction database presented in Table 1.3, which will serve as a running example. In this scenario, the item set *I* consists of $\{x_1, x_2, x_3, x_4, x_5\}$, representing various products sold in a retail store, such as *milk, bread, chicken, chocolates and coffee*. The given database in Table 1.3 comprises five transactions (T_1, T_2, T_3, T_4, T_5). For instance, transaction T_4 indicates the purchase of items x_1, x_3, and x_5, with respective purchase quantities (internal utilities) of 2, 6, and 2. External utilities, denoting the unit profits of the items, are provided in Table 1.4. In this case, the profit obtained from selling one unit of items x_1, x_2, x_3, x_4 and x_5 is \$5, \$2, \$1, \$2, and \$3, respectively.

Table 1.4. An example for external utility of item

Item	External utility
x_1	5
x_2	2
x_3	1
x_4	2
x_5	3

The objective of high utility itemset mining is to identify sets of items, known as itemsets, that are present in a quantitative database and exhibit high utility, such as generating substantial profits. The utility of an itemset serves as a gauge of its significance within the database and is calculated using a utility function. While the utility measure is typically defined as stated below, alternative measures have also been proposed [19]. In the given example, the utility measure is understood as the profit amount generated by each set of items.

Definition 4 (*Utility value*)

- The utility of item *i* in transaction T_k is denoted and calculated as
$$u(i, T_k) = p(i) \times q(i, T_k)$$
where $q(i, T_k)$ is the internal utility of item i in transaction T_k and $p(i)$ is the external utility of the item
- The utility of itemset *X* in transaction T_k is denoted and calculated as
$$u(X, T_k) = \sum_{i \in X} u(i, T_k)$$
- The utility of itemset *X* in transaction database *D* is denoted and calculated as
$$u(X) = \sum_{T_k \in D} u(X, T_k)$$

For instance, let's consider item x_1 in transaction T_4. The utility of x_1 is calculated as $u(x_1, T_4) = 5 \times 2 = 10$. Now, let's examine the utility of the itemset $\{x_1, x_3\}$ in T_4. It can be calculated as $u(\{x_1, x_3\}, T_4) = u(x_1, T_4) + u(x_3, T_4) = 5 \times 2 + 1 \times 6 = 16$. Moving to the utility of the itemset $\{x_1, x_3\}$ in the entire database, denoted as $u(\{x_1, x_3\})$, it is determined by adding up the individual utilities of x_1 and x_3 across all relevant transactions. Therefore, $u(\{x_1, x_3\}) = u(x_1) + u(x_3) = u(x_1, T_1) + u(x_1, T_3) + u(x_1, T_4) + u(x_3, T_1) + u(x_3, T_3) + u(x_3, T_4) = 5 + 5 + 5 + 10 + 1 + 1 + 6 = 28$

Consequently, the utility of $\{x_1, x_3\}$ in the database can be interpreted as the total profit generated when both items a and c are purchased together. The problem of high utility itemset mining can be defined as follows:

Definition 5 (*High-utility itemset*)

An itemset *X* is considered a *high-utility itemset* when its utility, denoted as *u(X)*, exceeds a minimum utility threshold specified by the user (i.e., *u(X)* ≥ *minutil*). If the utility of *X* falls below this threshold, *X* is classified as a *low-utility itemset*.

Definition 6 (*High-utility itemset mining problem*)

The objective of *high-utility itemsets mining* is to identify and uncover all itemsets with significant utility, based on a user-defined minimum utility threshold (*minutil*) [19].

It should be noted that in certain research studies, the utility of an itemset is presented as a proportion or percentage of the overall utility in the database. This approach, known as absolute utility [26], is equivalent to the definition mentioned earlier and yields the same patterns.

High utility itemset mining has a wide range of applications. In the context of market basket analysis, the high-utility itemset mining problem can be understood as identifying all item sets that have generated profits equal to or greater than the *minutil*. For instance, in our example, if *minutil* is set to 25, the set of High Utility Itemsets (HUIs) is presented in Table 1.5. Numerous algorithms have been proposed to discover high utility itemsets, which will be discussed in the following section.

Table 1.5. List of high utility itemsets satisfy *minutil* = 25

Itemset	Utility
$\{x_1, x_3\}$	28
$\{x_1, x_3, x_5\}$	31
$\{x_1x_2, x_3, x_4$	25
$\{x_2, x_3\}$	28
$\{x_2, x_3, x_4\}$	34
$\{x_2, x_3, x_4, x$	40
$\{x_2, x_3, x_5\}$	37
$\{x_2, x_4\}$	30
$\{x_2, x_4, x_5\}$	36
$\{x_2, x_5\}$	31
$\{x_3, x_5\}$	27

A noteworthy observation is that the problem of mining high utility itemsets encompasses the problem of mining frequent itemsets, making any algorithm designed for high utility itemset discovery applicable to frequent itemset mining in a transaction database as well. To achieve this, the following steps can be followed:

1. The transaction database is transformed into a quantitative transaction database. In this process, every item *i* in the item set *I* is assigned an external utility value of 1, denoted as *p(i)* = 1, to signify equal importance among all items. Additionally, for each item *i* and transaction T_c, if *i* is present in T_c, *q(i, T_c)* is set to 1. On the other hand, if *i* is not found in T_c, *q(i, T_c)* is set to 0.
2. Next, the quantitative transaction database obtained is subjected to a high utility mining algorithm using the *minutil* set to the value of *minsup*. This process aims to extract the frequent itemsets from the database.

For instance, the database presented in Table 1.1 can be converted into a quantitative database, resulting in the transaction databases shown in Tables 1.6 and 1.7. Subsequently, frequent itemsets can be extracted from this database by employing a high utility itemset mining algorithm. However, while a high utility itemset mining algorithm can be used to mine frequent itemsets, it may be more advantageous to utilize frequent itemset mining algorithms in situations where performance is crucial, as these algorithms are specifically optimized for this task.

Table 1.6. Quantitative transaction database corresponds to the database depicted in Table 1.1

TID	Transaction
T_1	$(x_1, 1), (x_2, 1), (x_3, 1), (x_4, 1), (x_5, 1)$
T_2	$(x_2, 1), (x_3, 1), (x_4, 1), (x_5, 1)$
T_3	$(x_1, 1), (x_3, 1), (x_4, 1)$
T_4	$(x_1, 1), (x_3, 1), (x_4, 1)$
T_5	$(x_2, 1), (x_3, 1), (x_5, 1)$

Table 1.7. The external utility values associated with the database presented in Table 1.6

Item	External utility
x_1	1
x_2	1
x_3	1
x_4	1
x_5	1

1.2.3 Key Properties of the Problem of High Utility Itemset Mining

The problem of mining high utility itemsets always has a unique solution, which involves listing all patterns from a given quantitative database that have a utility value higher than or equal to the minimum utility threshold specified by the user.

The difficulty in high utility itemset mining arises from two main factors. Firstly, the sheer number of itemsets that need to be considered can be extremely large in order to identify those with high utility. In general, if a database contains *m* distinct items, there are $2^m - 1$ possible itemsets (excluding the empty set). For instance, if we have $I = \{x_1, x_2, x_3\}$, the possible itemsets would be $\{x_1\}$, $\{x_2\}$, $\{x_3\}$, $\{x_1, x_2\}$, $\{x_1, x_3\}$, $\{x_2, x_3\}$, and $\{x_1, x_2, x_3\}$, resulting in a total of $2^3 - 1 = 7$ itemsets. As a result, the number of itemsets can grow exponentially, making it impractical to count the utilities of all possible itemsets by scanning the entire database.

The inefficiency of the naive approach becomes evident when dealing with a large number of items, such as in a retail store with 10,000 items (m = 10,000). In such cases, the naive approach would require calculating the utilities for $2^{10,000} - 1$ possible itemsets, which is unmanageable. Notably, even small databases can pose challenges in high utility itemset mining. For example, a database with a single transaction of 100 items would result in $2^{100} - 1$ possible itemsets. Hence, the search space, or the number of possible itemsets, can be substantial even when the number of transactions is low. The size of the search space is influenced by factors such as the database's transaction similarity, utility values, and the user's chosen minimum utility threshold.

Another reason that contributes to the difficulty of high utility itemset mining is the scattered distribution of high utility itemsets within the search space. Consequently, an algorithm must evaluate numerous itemsets before identifying the actual high utility itemsets. To visually demonstrate this point, Figure 1.1 presents a *Hasse diagram* as a graphical representation of the search space using a running example. In Hasse diagram, each potential itemset is depicted as a node, beside that an arrow connects an itemset *X* to another itemset *Y* if and only if $X \subseteq Y$ and $|X| + 1 = |Y|$. In Figure 1.1, light gray nodes depict high utility itemsets, while white nodes represent low utility itemsets. The diagram also provides the utility value for each itemset. A significant insight from the depicted figure is that the utility of an itemset can be greater, equal or even lower to the utility of any of its supersets or subsets. For instance, the itemset $\{x_2, x_3\}$ has the utility value of 28, whereas the utility of its supersets $\{x_2, x_3, x_4\}$ is 34 and $\{x_1, x_2, x_3, x_4, x_5\}$ is 25. This observation leads to the conclusion that the utility measure does not exhibit a monotone or anti-monotone behaviour.

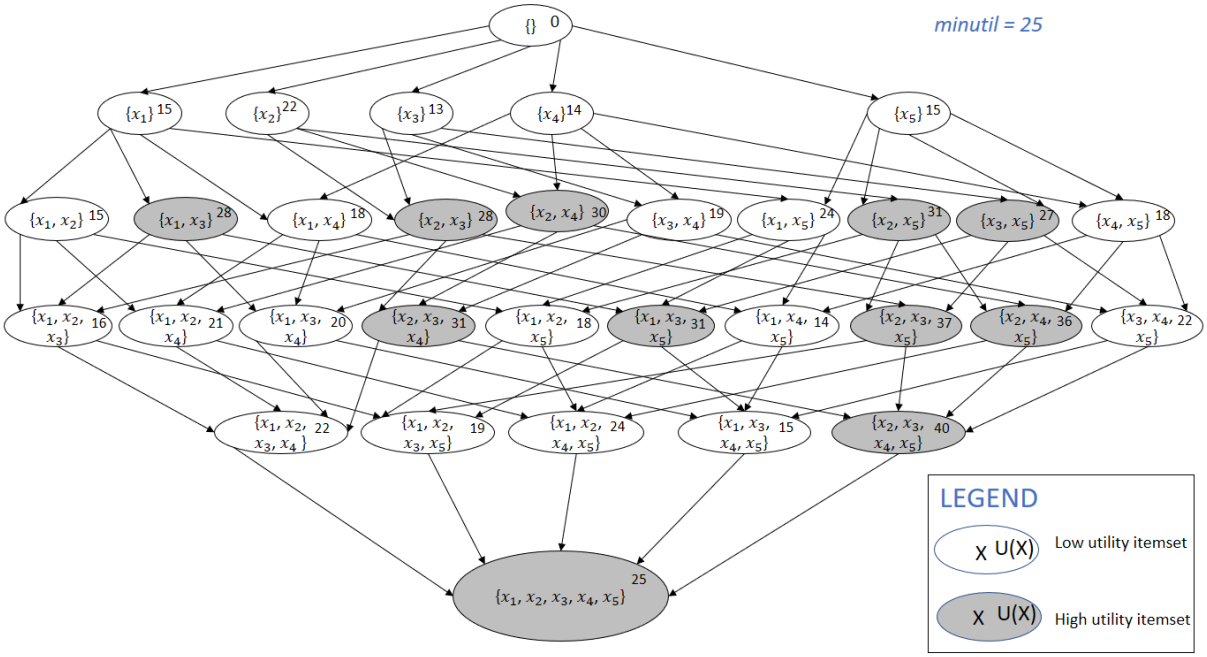

Figure 1.1. An example for search space of high utility itemset mining

Property 1. The utility measure is not characterized by either monotonicity or anti-monotonicity.

When considering two itemsets, X and Y, where X is a subset of Y ($X \subset Y$), their utility relationship can fall into one of three possibilities: the utility of X is less than the utility of Y ($u(X) < u(Y)$), the utility of X is greater than the utility of Y ($u(X) > u(Y)$), or the utilities of X and Y are equal ($u(X) = u(Y)$) [19].

Due to this characteristic, the high utility itemsets are dispersed throughout the search space, as evident in Figure 1.1. This is the primary factor that makes high utility itemset mining more challenging compared to frequent itemset mining. When dealing with frequent itemset mining problem, the support measure exhibits the desirable property of monotonicity [5]. This means that the support of an itemset is always greater than or equal to the frequency of any of its supersets.

Property 2. The support measure is monotonic.

When considering two itemsets, X and Y, where X is a subset of Y ($X \subset Y$), it can be concluded that the support of X is greater than or equal to the support of Y ($\sup(X) \geq \sup(Y)$).

For instance, consider the database provided in Table 1.1, where the support of $\{x_2, x_3\}$ is 3, and the support of its supersets $\{x_2, x_3, x_4\}$ and $\{x_1, x_2, x_3, x_4, x_5\}$ is 2 and 1, respectively. The support measure's monotonicity simplifies the identification of frequent patterns by ensuring that if an itemset is infrequent, all of its supersets are also infrequent. This property allows a frequent itemset mining algorithm to eliminate all supersets of an infrequent itemset from the search space. As an illustration, when the algorithm identifies that the itemset $\{x_1, x_4\}$ is infrequent, it can immediately eliminate all supersets of $\{x_1, x_4\}$, resulting in a substantial reduction in the search space. The visualization of the search space in Figure 1.2, based on the example database in Table 1.1, clearly depicts a distinct boundary between frequent itemsets and infrequent itemsets, demonstrating the anti-monotonic characteristic of the support measure. Property 2, referred to as the *anti-monotonicity property*, *Apriori property* or *downward-closure property* represents this particular feature. Although it applies to the support measure, it does not extend to the utility measure used in high utility itemset mining. Consequently, Figure 1.1 does not facilitate a clear differentiation between low utility itemsets and high utility itemsets.

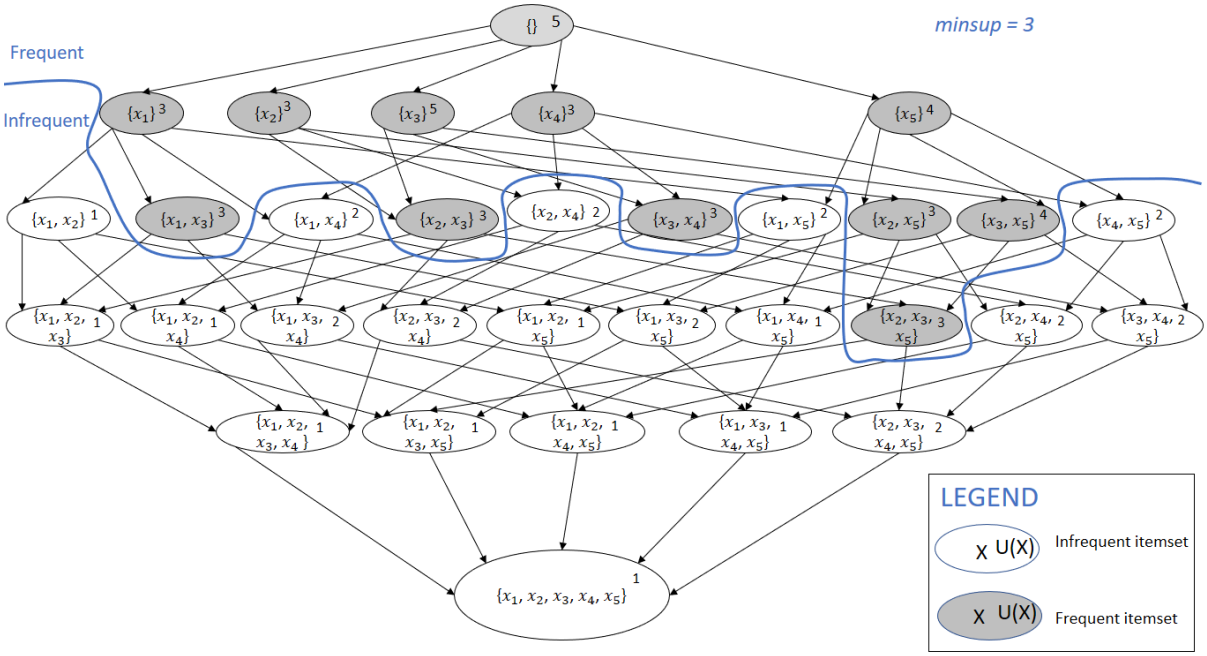

Figure 1.2. Demonstrate search space when using data in Table 1.1 to solve frequent itemset mining

Given the extensive search space involved in high utility itemset mining, it is crucial to develop rapid algorithms that can circumvent the necessity of examining every possible itemset. These algorithms should process each itemset within the search space with optimal efficiency while ensuring the identification of all high utility itemsets. Furthermore, since the utility measure lacks monotonicity and anti-monotonicity, effective strategies employed to reduce the search space in frequent itemset mining cannot be directly applied to address the challenges of high utility itemset mining.

1.2.4 Correlated High Utility Itemset Mining

One drawback of traditional algorithms used for mining high utility itemsets is their tendency to uncover itemsets with high profitability that consist of weakly correlated items. Such itemsets can be misleading or unhelpful for making marketing decisions. For instance, let's consider a transaction database from a retail store. Existing algorithms might identify the purchase of a high-end smartphone and a pack of chewing gum as a high-utility itemset due to the overall profitability when these two items are sold together. However, it would be a mistake to utilize this pattern to promote high-end smartphones to customers who buy chewing gum because, upon closer examination, it is evident that these two items are rarely purchased together. The explanation for why this pattern can still be classified as a high utility itemset, despite the minimal correlation between high-end smartphones and chewing gum, is due to the high cost of the smartphones. As a result, almost any item combined with a high-end smartphone has the potential to be considered a high utility item. The significance of this drawback in conventional high utility itemset mining algorithms cannot be overlooked. A comprehensive experimental study demonstrated that a significant majority, less than 1%, of the patterns generated by traditional high utility itemset mining algorithms exhibit strong correlations among the items.

Different measures have been utilized to gauge the correlation between items within an itemset, such as the bond [27], all-confidence [28], and affinity [29, 30] measures, aiming to address this issue. The general problem of extracting correlated high utility itemsets using the bond measure and all-confidence measure can be described as follows. The detail about

correlated high utility itemset mining algorithms using these measures will also be discussed in the next chapter.

Definition 7 The bond measure

Consider an itemset X. The disjunctive support of itemset *X* in database *D*, represented as *dissup(X),* is defined as follow:

$$|\{T_k \in D | X \cap T_k \neq \emptyset\}|$$

The *bond* measure of itemset X, denoted as *bond(X),* is calculated as:

$$bond(X) = \frac{sup(X)}{dissup(X)}$$

The bond measure falls within the interval [0,1] and demonstrates monotonicity [31].

Definition 8 Using the bond measure to find correlated high utility itemset

Given a quantitative transaction database and user-defined thresholds for minimum bond (*min_bond*) and *minutil*, the challenge in discovering correlated high utility itemsets using the bond measure is to identify and output all itemset X that satisfies both two conditions

- $bond(X) \geq min\_bond$
- $u(X) \geq minutil$

For example, when using a minimum utility threshold of 25 and a minimum bond threshold of 0.6, the correlated high utility itemsets are displayed in Table 1.8. Let's take the high utility itemset $\{x_1, x_2, x_3, x_4, x_5\}$ as an illustration. It is deemed uncorrelated due to its bond value of 0.2.

Table 1.8. Correlated high-utility itemsets satisfy *min_bond*=0.6 and *minutil*=25

Itemset	Utility	Bond value
$\{x_1, x_3\}$	1	0.6
$\{x_2, x_3\}$	1	0.6
$\{x_3, x_5\}$	1	0.8
$\{x_2, x_3, x_5\}$	1	0.6

In order to effectively identify correlated high-utility itemsets using the bond measure, the $FCHM_{bond}$ algorithm [27] was introduced as an extension of the FHM algorithm. Experimental findings have demonstrated that FCHM outperforms the FHM algorithm significantly by efficiently pruning a large number of weakly correlated high utility itemsets. Furthermore, an alternative correlation measure called *all-confidence* was integrated into the $FCHM_{all-confidence}$ algorithm [32]. The all-confidence measure is demonstrated as follows.

Definition 9 The all-confidence measure

The all-confidence measure of itemset X can be calculated as

$$all - confidence(X) = \frac{\sup(X)}{argmax\{\sup(Y) | \forall\, Y \subset X \wedge Y \neq \emptyset\}}$$

The all-confidence measure of an itemset is a numerical value that falls within the [0, 1] range, with a higher value indicating a stronger correlation between the items. For instance, consider the itemset $\{x_1, x_4\}$. The all-confidence of this itemset is computed as

$$all - confidence(\{x_1, x_4\}) = \frac{\sup(\{x_1, x_4\})}{argmax\{\sup(\{x_1\}), \sup(\{x_4\})\}} = \frac{2}{3} = 0.67$$

1.3 Related works and contribution

Correlated high utility itemset mining is a field that has received significant attention in recent years. Therefore, in addition to the FCHM algorithm, there have been numerous other algorithms and prune strategies proposed. Some of these works are described below:

Paper [33] is the first paper that use a null-invariant measure to find high utility itemsets. The author developed an algorithm named CoHUIM based on a project-based approach to reduce search space and memory usage. The measure used is Kulczynsky measure (Kulc). Although this measure does not have an anti-monotonicity property, the author has introduced another property that can also help prune candidates effectively which is the sorted downward closure property. Experimental results show that the algorithm returns a smaller number of itemsets but carries more valuable information to the user. Runtime and memory of this algorithm are also within acceptable thresholds.

In [34], the author proposes CoUPM algorithm to find itemsets that satisfy both utility threshold and correlation threshold using Kulc measure. The author has taken advantage of the downward closure property of Kulc measure to increase the performance of the algorithm. In addition, CoUPM also introduces revised utility list, which helps the algorithm does not need to go through the entire database several times. Experiments on many real-world datasets show that CoUPM works well in terms of both effectiveness and efficiency.

The author in [35] proposed a single phase algorithm named CoHUI-Miner to find correlated high utility patterns. This algorithm uses the database projection mechanism to reduce the database size. Beside that a new concept is introduced called the prefix utility of projected transactions. This new concept helps to effectively eliminate patterns which do not meet the minimum threshold during the mining process. Experiments on many datasets of different types show that the proposed algorithm has better performance than the CoHUIM algorithm [33].

Paper [36] introduces a novel pattern mining algorithm called CoHAI miner, which is based on correlation and average-utility. The algorithm aims to discover patterns with strong correlations and high average-utility. Unlike previous methods, CoHAI miner uses the correlation factor *Kulc* to ensure the presence of strong correlations in the patterns. The algorithm is designed as a single-phase process and utilizes a vertical list structure called SAU list for efficient pruning and average-utility calculation. Notably, the SAU list of an itemset can be constructed by combining the SAU lists of its subsets without the need for additional dataset scans. The author conducted comprehensive experiments to compare the performance of CoHAI miner with an existing algorithm called EHAUPM. The results indicate that CoHAI miner exhibits significant improvements in runtime and generates more meaningful patterns compared to EHAUPM.

From all above study, it can be seen that the common point of correlated high utility itemset mining algorithms is that the measure must satisfy some properties that support the process of pruning candidates. Besides, it is also necessary to apply suitable strategies for each different measure to improve the performance of the algorithm. Therefore, finding other measures that satisfy the above conditions will help expand the limit of usable measures, thereby facilitating the improvement of both effectiveness and efficiency of the correlated high utility itemset mining algorithms in general.

In this thesis, we propose a new version of the FCHM algorithm called $FCHM_{cosine}$. Specifically, we demonstrate that the cosine measure possesses the anti-monotonicity property, thereby proving its suitability for integration with the FCHM algorithm to create the $FCHM_{cosine}$ algorithm, similar to the previous integration of the bond measure and all-confidence measure.

Chapter 2. Algorithms and methodology

2.1 The FHM algorithm

To address the main challenge in high utility itemset mining, it is crucial to establish a monotonically increasing measure that serves as an upper-bound for the utility measure. This measure is used to effectively reduce the search space while ensuring that no high utility itemsets are overlooked. The proposed measure in this context is the TWU (Transaction Weighted Utilization) measure, which is defined as follows.

Definition 10 (TWU measure)

The *transaction utility (TU)* of a transaction T_c is calculated by summing up the utilities of all the items in T_c, denoted as $TU(T_c) = \sum_{x \in T_c} u(x, T_c)$ for each item *x* in T_c. On the other hand, the TWU of an itemset *X* is defined as the total sum of transaction utilities from transactions that contain *X*. Mathematically, $TWU(X) = \sum_{T_c \in g(X)} TU(T_c)$ for each transaction T_c in the set of transactions *g(X)* that contain *X*.

For example, consider the transaction utilities of T_1, T_2, T_3, T_4, and T_5, which are 25, 20, 8, 22, and 9, respectively. The TWU values for the individual items x_1, x_2, x_3, x_4 *and* x_5 are 55, 54, 84, 53, and 76, respectively. If we consider the itemset $\{x_3, x_4\}$, the TWU is calculated as $TWU(\{x_3, x_4\}) = TU(T_0) + TU(T_1) + TU(T_2) = 25 + 20 + 8 = 53$. It is important to note that the TWU measure serves as an upper-bound on the utility measure and exhibits monotonicity. This property formalizes the relationship between TWU and utility measures.

Property 3 (The TWU measure serves as an upper-bound on the utility measure and exhibits monotonicity)

For any itemset X, the TWU of X is equal to or greater than its utility (TWU(X) ≥ u(X)). Additionally, the TWU of X is equal to or greater than the utility of its supersets (TWU(X) ≥ u(Y) for all Y that are supersets of X). This property has been proven in [18]. Essentially, the TWU of X represents the sum of utilities in transactions where X is present, ensuring that it is at least as high as the utility of X itself and any of its supersets.

The TWU measure is useful for reducing the search space in itemset mining. To facilitate this, a specific property has been proposed.

Property 4 (Pruning the search space based on the TWU)

From Property 3, it follows that if the TWU of an itemset X is less than the minutil, then X and all its supersets are considered low-utility itemsets.

For instance, consider the itemset $\{x_1, x_2, x_3, x_4\}$ with a utility of 20 and a TWU of 25. According to Property 4, any supersets of $\{x_1, x_2, x_3, x_4\}$ cannot have a TWU or utility value greater than 25. Therefore, if the user sets the minimum *utility* threshold to a value higher than 25, all supersets of $\{x_1, x_2, x_3, x_4\}$ can be pruned from the search space since it is guaranteed by Property 4 that their utilities cannot exceed 25.

Another commonly used approaches for high utility itemset mining algorithms is the *utility-list* structure. This structure was initially introduced in the HUI-Miner algorithm [17], which is a generalized version of the *tid-list* structure [23] that is used for frequent itemset mining. Over time, several faster utility-list based algorithms have been proposed, including FHM [14], mHUIMiner [37], and ULB-Miner [38]. Additionally, extensions have been made to address different variations of the high utility itemset mining problem. One of the reasons for the popularity of utility-list based algorithms is their speed and ease of implementation. In this subsection, we focus on the FHM algorithm, which is a representative utility-list based algorithm that has been shown to be up to seven times faster than HUI-Miner. This algorithm has been widely used and extended by many researchers.

The FHM algorithm is a one-phase algorithm that utilizes a depth-first search to explore the search space of itemsets. As the algorithm traverses the search space, it generates a *utility-*

list for each visited itemset. The utility-list for an itemset contains information regarding the utility of the itemset in transactions where it occurs, as well as information about the utilities of the remaining items in those transactions. Utility-lists enable quick calculation of the utility of an itemset and provide upper-bounds on the utility of its super-sets without the need to scan the entire database. Additionally, utility-lists for *k*-itemsets ($k > 1$) can be efficiently generated by combining the utility-lists of shorter patterns. The utility-list structure is formally defined as follows:

Definition 11 (*Utility-list*)

Let *X* be an itemset and *D* be a quantitative database. It is assumed, without loss of generality, that a total order, denoted by $\succ$, is defined on the set of items I appearing in the database. The *utility-list ul(X)* for *X* in a quantitative database *D* is a set of tuples. For each transaction T_{tid} containing *X*, there is a tuple (*tid, iutil, rutil*). The *iutil* element of the tuple represents the utility of *X* in T_{tid}, i.e., *u(X, T_{tid})*. The *rutil* element of the tuple is defined as the summation of the utilities of all items i in T_{tid} that appear before any item in *X* in the total order defined by $\sum_{i \in T_{tid} \wedge i \succ x \forall x \in X} u(i, T_{tid})$

Assuming a total order $\succ$ is defined on the set of items in a quantitative database D, the utility-lists of three itemsets $\{x_1\}$, $\{x_4\}$, and $\{x_1, x_4\}$ are presented in Figure 2.1 as an example. The utility-list of $\{x_1\}$ contains three tuples corresponding to transactions T_1, T_3, and T_4, where $\{x_1\}$ appears. The second column of the utility-list (*iutil* values) of $\{x_1\}$ displays that the utility of $\{x_1\}$ in T_1, T_3, and T_4, is 5, 5, and 10, respectively. The third column of the utility-list of $\{x_1\}$ presents that the *rutil* values of $\{x_1\}$ for transactions T_1, T_3, and T_4 are 20, 3, and 10, respectively.

The utility list of $\{x_1\}$

tid	iutil	rutil
T_0	5	20
T_2	5	3
T_3	10	12

The utility list of $\{x_4\}$

tid	iutil	rutil
T_0	6	3
T_1	6	3
T_2	2	0

The utility list of $\{x_1, x_4\}$

tid	iutil	rutil
T_0	11	3
T_2	7	0

Figure 2.1. Construct utility-list of $\{x_1, x_4\}$ from utility list of $\{x_1\}$ and $\{x_4\}$

The FHM algorithm creates the utility-lists for single items by scanning the database once. To create utility-lists for larger itemsets, it joins the utility-lists of smaller itemsets. The join operation for single items involves creating a tuple for each pair of tuples in the utility-lists of *x* and *y*, where *x* and *y* are items such that *x* comes before *y* in the total order. The utility-list of a larger itemset $P \cup \{x, y\}$, where *x* and *y* are items such that *x* comes before *y* in the total order, is created by subtracting the utility of P from the sum of the utilities of *x* and *y* in each tuple that has the same transaction ID in the utility-lists of *x*, *y*, and *P*. This can be done without scanning the database, for example, the utility-list of $\{x_1, x_4\}$ can be obtained by joining the utility-lists of $\{x_1\}$ and $\{x_4\}$ shown in Figure 2.1.

The utility-list structure is a valuable tool because it enables the retrieval of an itemset's utility without having to search the entire database.

Property 5 (Determining the utility of an itemset by utilizing the utility-list structure associated with that itemset). *To calculate the utility of an itemset X, you can simply add up the iutil values in its utility-list ul(X). This means that the utility of X is equal to the sum of all the iutil values in ul(X)* [17]. *In mathematical terms, this can be expressed as: u(X) =* $\sum_{e \in ul(X)} e.iutil$

The utility of an itemset can be calculated by summing the values in the *iutil* column of its utility-list. For example, the utility of $\{x_1, x_4\}$ can be found by adding the values 11 and 7 from its utility-list. The utility-list is also helpful in reducing the search space by using the minimum utility threshold and downward closure property. The minimum utility threshold is the

minimum utility that an itemset must have to be considered, while the downward closure property ensures that all of an itemset's subsets are also considered if the itemset itself is considered.

Definition 12 (*Remaining utility upper-bound*)

The *"remaining utility upper-bound"* for an itemset X is the sum of its *"iutil"* and *"rutil"* values in its *"utility-list"* called *"ul(X)"*. It is calculated by adding up the *iutil* and *rutil* values for each transaction containing *X*. The purpose of this upper-bound is to limit the search space when mining frequent itemsets. It serves as an upper-bound on the utility of *X* and all its extensions, meaning that any itemset *Y* that is an extension of *X* has a utility value less than or equal to the *reu(X)* value.

To illustrate, consider the example of calculating the remaining utility upper-bound of the itemset $\{x_1, x_4\}$ using its utility-list (shown in Figure 2.1). The upper-bound is the sum of the *iutil* and *rutil* values in its utility-list, i.e., *reu*($\{x_1, x_4\}$) = 11 + 7 = 18. Therefore, it can be inferred that the itemset $\{x_1, x_4\}$ and all its extensions, such as $\{x_1, x_4, x_5\}$, cannot have a utility greater than 18. When *minutil* = 25, as in the running example, these itemsets can be pruned from the search space, as they are considered low-utility itemsets. This is expressed by the following property.

Property 6 (Pruning search space based on utility-list)

To prune the search space, we can use the utility-list of an itemset X. If the sum of its iutil and rutil values in ul(X) is less than the minutil value, which is the minimum utility threshold, then X and all its extensions are considered low utility itemsets. In other words, if reu(X) < minutil, X and its extensions will not meet the minimum utility threshold and can be pruned from the search space. [17]

The FHM algorithm (Algorithm 1) takes a quantitative transaction database and a minimum utility threshold as input. Firstly, FHM scans the database to calculate the TWU of each item. Then, it identifies a set of all items having a TWU no less than the minimum utility threshold, which is named $I*$. Items with TWU less than the minimum threshold are ignored, as per Property 4. Items are then ordered based on their ascending TWU values, which is used to create a total order on items called the $\succ$ order. During a database scan, items in transactions are reordered based on the $\succ$ order, and the utility-list of each item in I$*$ is constructed. A structure called the EUCS (Estimated Utility Co-Occurrence Structure) is then built as a set of triples that indicate the TWU of item pairs. The EUCS is implemented as a triangular matrix and occupies a small amount of memory. The FHM algorithm uses the EUCS to prune the search space. Finally, the depth-first search of itemsets starts by calling the recursive procedure FHM Search with the empty itemset ∅, *I*$*$, *minutil*, and the EUCS structure. More details about the construction of the EUCS and implementation optimizations can be found in the paper about FHM [14].

Algorithm 1: The FHM algorithm

---

input: D: a transaction database, *minutil*: a user-specified threshold
output: the set of high-utility itemsets

1 Scan D to calculate the TWU of single items;
2 *I** ← each item *i* such that TWU(*i*) ≥ *minutil*;
3 Let ≻ be the total order of TWU ascending values on *I**;
4 Scan D to built the utility-list of each item *i* ∈ *I** and build the *EUCS*;
5 Output each item *i* ∈ *I** such that SUM({*i*}.*utilitylist.iutils*) ≥ *minutil*;
6 FHMSearch(∅, *I**, *minutil*, *EUCS*);

Algorithm 2: The *FHMSearch* procedure

input: *P*: an itemset, *ExtensionOfP*: a set of extensions of *P*, *minutil*: a user-specified threshold, *EUCS*: the EUCS structure
output: the set of high-utility itemsets

1 foreach *itemset* $P_x \in$ ExtensionOfP **do**
2 if SUM(P_x.*utilitylist.iutils*)+SUM(P_x.*utilitylist.rutils*) ≥ *minutil* **then**
3 ExtensionOf$P_x \leftarrow \emptyset$;
4 **foreach** *itemset* $P_Y \in$ ExtensionOfP *such th*at y ≻ x **do**
5 **if** ∃(*x, y, c*) ∈ EUCS such that c ≥ *minutil* **then**
6 $P_{xy} \leftarrow P_x \cup P_y$;
7 P_{xy}.utilitylist ← Construct(*P*, P_x, P_y);
8 $ExtensionOfP_x \leftarrow ExtensionOfP_x \cup P_{xy}$;
9 **if** SUM(P_{xy}.*utilitylist.iutils*) ≥ *minutil* **then** output P_x
10 **end**
11 **end**
12 FHMSearch(P_x, $ExtensionOfP_x$, *minutil*);
13 end
14 end

Algorithm 3: The Construct procedure

input: P: an itemset, P_x: the extension of *P* with an itemset *x*, P_y: the extension of *P* with an item *y*
output: the utility-list of P_{xy}

1 $UtilityListOfP_{xy} \leftarrow \emptyset$;
2 foreach *tuple ex* ∈ P_x.*utilitylist* **do**
3 **if** ∃ *ey* ∈ P_y.*utilitylist* and *ex.tid* = *exy.tid* **then**
4 **if** *P.utilitylist* ≠ ∅ **then**
5 Search element *e* ∈ *P.utilitylist* such that *e.tid* = *ex.tid*;
6 *exy* ← (*ex.tid*, *ex.iutil* + *ey.iutil*, *ey.rutil*)
7 **end**
8 **else** *exy* ← (*ex.tid*, *ex.iutil* + *ey.iutil*, *ey.rutil*)
$UtilityListOfP_{xy} \leftarrow UtilityListOfP_{xy} \cup \{exy\}$;
9 **end**
10 end
11 return $UtilityListOfP_{xy}$;

The *FHM Search* algorithm (Algorithm 2) receives as input an itemset *P*, its extensions in the form of P_z where *z* is appended to *P*, a minimum utility value, and the EUCS. The algorithm starts by examining each extension P_x of *P*. If the sum of the *iutil* values in the utility-list of P_x is greater than or equal to *minutil*, then P_x is considered a high-utility itemset and is output (as per Property 4). If the sum of *iutil* and *rutil* values in the utility-list of P_x is greater than or equal to *minutil*, then the algorithm merges P_x with all extensions P_y of *P* such that $y \succ x$ to create extensions of the form P_{xy}, which contain $|P_x| + 1$ items. The utility-list of P_{xy} is obtained by

calling the Construct procedure to join the utility-lists of P, P_x, and P_y. The algorithm then performs a recursive call to the *Search* procedure with P_{xy} to calculate its utility and explore its extension(s). The *FHM Search* algorithm recursively explores the search space of itemsets by appending single items and only prunes the search space based on Property 6. It can be proven that this procedure is correct and complete in discovering all high-utility itemsets.

Item	x_1	x_2	x_3	x_4
x_2	25			
x_3	55	54		
x_4	33	45	53	
x_5	47	54	76	45

Figure 2.2. The co-occurrence structure for estimated-utility

The FHM algorithm utilizes a *vertical database representation* in the form of a utility-list structure. This structure provides a list of transactions where each itemset appears, which is different from a horizontal database representation where each entry is a transaction showing the items it contains.

One of the advantages of utility-list based algorithms is that they are simple to implement and are efficient. In comparison to two-phase algorithms, utility-list based algorithms can be more than two orders of magnitude faster [14, 17, 21]. However, these algorithms have some significant drawbacks. Firstly, since these algorithms create itemsets by combining other itemsets and not by reading the database, they may explore itemsets that do not actually exist in the database, leading to the wastage of time in constructing utility-lists. Secondly, these algorithms can consume a lot of memory because they need to build a utility-list for each itemset in the search space. The utility-list of an itemset can be quite large, and in the worst-case scenario, it can contain a tuple for every transaction in the database. Additionally, the join operation can be expensive, as it requires the comparison of two or three utility-lists to construct the utility-list of each *k*-itemset ($k > 1$).

To decrease the memory usage of utility-list based algorithms, a new algorithm called ULB-Miner [38] has been developed by expanding upon the HUI-Miner and FHM algorithms. ULB-Miner uses a buffer to conserve memory used for storing utility-lists. This approach has demonstrated an improvement in both runtime and memory usage. Another enhancement to HUI-Miner is HUI-Miner* [39], which utilizes an improved utility list* structure to accelerate HUI-Miner.

2.2 The FCHM algorithm

As mentioned in the previous chapter, traditional high utility itemset mining algorithms may have a drawback of discovering itemsets with high profitability but containing weakly related items. Such itemsets may be misleading or not useful for making marketing decisions. For example, in a retail store's transaction database, an algorithm may identify the purchase of a 50-inch plasma television and a pen as a high-utility itemset due to high profitability when sold together. However, this pattern may not be suitable for promoting plasma televisions to pen buyers as these two items are rarely sold together. The reason for this is that expensive items such as plasma televisions can lead to almost any items combined with them being considered a high utility itemset, even if there is a low correlation between them. Traditional high utility itemset mining algorithms are limited in that they often discover itemsets with high profits that include weakly correlated items, making such itemsets misleading or unhelpful for making marketing decisions. Empirical evidence shows that in many cases, less than 1% of the patterns discovered by these algorithms contain strongly correlated items. Different measures have been proposed to deal with this issue of traditional high utility itemset mining algorithms. These measures include the bond [27], all-confidence [28], and affinity measures [29, 30]. They are used to measure the level of correlation between items in an itemset.

Algorithm 4: The FCHM algorithm

---

input: D: a transaction database, *minutil*: a user-specified threshold,
min_measure: a user-specified threshold for all-confidence or
bond measure
output: the set of correlated high-utility itemsets

1 Scan D to calculate the TWU of single items;
2 $I^* \leftarrow$ each item i such that TWU(i) $\geq$ *minutil*;
3 Let $\succ$ be the total order of TWU ascending values on I^*;
4 Scan D to built the utility-list of each item $i \in I^*$ and build the *EUCS*;
5 Output each item $i \in I^*$ such that SUM($\{i\}$.*utilitylist.iutils*) $\geq$ *minutil*;
6 FHMSearch($\emptyset$, I^*, *minutil*, *EUCS*, *min_measure*);

Algorithm 5: The *FCHMSearch* procedure

input: *P*: an itemset, *ExtensionOfP*: a set of extensions of *P*, *minutil*: a user-specified threshold, *EUCS*: the EUCS structure, min_measure: a user-specified threshold for all-confidence or bond measure
output: the set of high-utility itemsets

1 foreach *itemset* P_x thuoc ExtensionOfP **do**
2 if SUM(P_x.*utilitylist.iutils*)+SUM(P_x.*utilitylist.rutils*) ≥ *minutil* **then**
3 ExtensionOfP_x ← ∅;
4 **foreach** *itemset* P_Y ∈ ExtensionOfP *such th*at y ≻ x **do**
5 **if** ∃(*x*, *y*, *c*) ∈ EUCS such that c ≥ *minutil* **then**
6 P_{xy} ← P_x ∪ P_y;
7 P_{xy}.*utilitylist* ← Construct(*P*, P_x, P_y);
8 **if** P_{xy}.measure ≥ min_measure **then**
9 *ExtensionOf*P_x ← *ExtensionOf*P_x ∪ P_{xy};
10 **if** SUM(P_{xy}.*utilitylist.iutils*) ≥ *minutil* **then** output P_x
11 **end**
12 **end**
13 **end**
14 FCHMSearch(P_x, *ExtensionOf*P_x, *minutil*);
15 end
16 end

Property 7 (Anti-monotonicity with the bond measures, all-confident).
If X and Y are two itemsets such that X is a subset of Y, then the bond value of X is always greater than or equal to the bond value of Y and the all-confidence value of X is always greater than or equal to the all-confidence value of Y.

2.2.1 The FCHM$_{all-confidence}$ Algorithm

The all-confidence of an itemset *X* can be calculated by dividing its support with the maximum support of its subsets, which is always equal to the support of an item in *X*. In order to calculate the all-confidence of *X*, it is necessary to find the support of *X* and its items. The support of *X* can be obtained by determining the size of its utility list, which is created at line 4 of Algorithm 4. The support of single items can be obtained from their respective utility lists. With this information, it is simple to find the maximum support of the subsets of an itemset *X* and calculate its all-confidence. Specifically, the all-confidence of an itemset *X* can be calculated as |*utility-list(X)*| divided by argmax{|*utility-list(i)*|} for all *i* in *X*. For instance, the all-confidence of the itemset *ab* can be determined as |*utility-list(ab)*| divided by *argmax*{*utility-list(a), utility-list(b)*}.

The process of finding the maximum support of items in an itemset X involves |*X*|-1 comparisons, which can be time-consuming. One way to reduce the number of comparisons is to store the maximum support of subsets of each itemset in its utility-list. Then, when constructing the utility-list of an itemset P_{xy}, the maximum support of its subsets can be obtained with just one comparison by using the formula *maxSubset(* P_{xy} *)* = *argmax{maxSubset(* P_x *), maxSubset(* P_y *)}*. To further improve the performance of FCHM$_{all-confidence}$, three strategies are presented in the following paragraphs for more efficiently discovering CHIs.

Strategy 1. Directly Ouputting Single items (DOS). High utility itemsets that consist of only one item are immediately returned as output because they have an all-confidence value of 1.
Strategy 2. Pruning Supersets of Non correlated itemsets (PSN). If the all-confidence of an itemset P_{xy} is lower than a minimum threshold value, then according to the anti-monotonic property of the all-confidence measure (Property 7), there is no need to explore any further extensions of P_{xy}.
Strategy 3. Pruning with Upper-Bound (PUB) version 1. The third approach aims to avoid constructing the utility-list of an itemset P_{xy}. and pruning all its extensions. To achieve this, a novel structure called *Support Matrix* is created during the second scan of the database. This structure stores the support of all itemsets that contain two items from the set $I*$, and its design is similar to that of the EUCS structure. The Support Matrix is defined formally as follows. The Support Matrix is a collection of triples in the form (*a*, *b*, *c*) where *a* and *b* are from the set $I*$, and *c* is a real number indicating the support of the itemset $\{a, b\}$. The Support Matrix only stores the triples where *c* is not equal to 0. For example, the Support Matrix for the given example is shown in Figure 2.3. To improve the $\text{FCHM}_{all-confidence}$ algorithm, Line 5 of Algorithm 4 is modified to include a condition that the utility-list of P_{xy}. is only created if:

$$\frac{Min\{support(P_x), support(P_y), support(xy)\}}{Max\{maxSubset(P_x), maxSubset(P_y)\}} \geq min\_measure$$

This pruning technique ensures the accuracy and completeness of the FCHM algorithm.

Item	x_1	x_2	x_3	x_4
x_2	1			
x_3	3	3		
x_4	2	2	3	
x_5	2	3	4	2

Figure 2.3. Matrix of support value

2.2.2 The FCHM_{bond} Algorithm

To calculate the bond of a high utility itemset, a simple but inefficient approach is to scan the database for each high utility itemset and calculate its support and disjunctive support. However, in FCHM, a more efficient approach is used. This involves appending a structure called *disjunctive bit vector* [31] to each utility-list, which can quickly calculate the disjunctive support of any itemset.

The bond measure can be computed inefficiently by scanning the database for each high utility itemset and calculating its support and disjunctive support. However, FCHM uses a more efficient approach by appending a *disjunctive bit vector* to each utility-list, denoted as *bv(X)*, which contains |*D*| bits. If the *j*-th bit is set to 1, it means that there exists an item *i* in *X* such that *i* is in the transaction T_j. Otherwise, it is set to 0. The disjunctive bit vector of each item is created during the first database scan. Then, the disjunctive bit vector of any larger itemset P_{xy} explored by the search procedure can be obtained efficiently by performing the logical OR operation between the bit vectors of P_x and P_y. The cardinality of the disjunctive bit vector of an itemset P_{xy} is equal to its disjunctive support, while the cardinality of its utility-list is equal to its conjunctive support. Thus, the bond of an itemset can be computed by dividing the cardinality of its utility-list by the cardinality of its disjunctive bit vector, which is obtained using the utility-list and the disjunctive bit vector.

The property mentioned earlier can be used to calculate the bond of any itemset produced by the search process after constructing its utility-list. This leads to the $FCHM_{bond}$ algorithm, which is accurate and thorough in finding CHIs. To further enhance FCHM's efficiency, the following paragraphs introduce additional methods for discovering CHIs. $FCHM_{bond}$ implements Strategies 1 and 2 from $FCHM_{all-confident}$, adjusts Strategy 3, and introduces three methods called Strategies 4, 5 and 6.

Strategy 3. Pruning with the Upper-Bound (PUB) version 2.

This approach is similar to the PUB version 1 strategy and uses the Support Matrix. In $FCHM_{bond}$, a modification is made to Line 5 of Algorithm 4 to include a condition that the utility-list of P_{xy} is constructed only if

$$\frac{Max\{support(P_x), support(P_y), support(xy)\}}{Min\{dissup(P_x), dissup(P_y), dissup(xy)\}} \geq min\_measure$$

Here, $dissup(xy) = support(x) + support(y) - support(xy)$, and $support(xy)$ is obtained from the Support Matrix. It is evident that this pruning condition maintains the accuracy and completeness of FCHM because the left-hand side of the inequality provides an upper-bound on the *bond*(P_{xy}).

Strategy 4. Abandoning Utility-List construction early (AUL). The fourth strategy proposed in FCHM is to halt the process of building the utility-list of an itemset when a particular condition is satisfied, implying that the itemset is unlikely to be a correlated high utility itemset. This approach is founded on a new discovery:

Property 8 (conjunctive support of an extension P_{xy})

An itemset P_{xy} can only be considered as correlated if its support is greater than or equal to the lower bound value of *lowerBound*(P_{xy}), which is calculated as the product of the minimum measure and the cardinality of the disjunctive support of P_{xy}, then take the ceiling function of the result just found.

The property that an itemset P_{xy} is correlated only if its support is no less than the value *lowerBound*(P_{xy}), can be directly derived from the bond measure definition. To construct the utility-list of itemset P_{xy} in the Construct procedure (Algorithm 3), some modifications are made. First, the disjunctive bit vector of P_{xy} is obtained by performing the OR operation with the bit vectors of P_x and P_y. This helps obtain the value of *support*(P_{xy}). Second, a variable called *maxSupport* is initialized to the conjunctive support of P_x. Third, the utility-list of P_{xy} is constructed by checking if each tuple in the utility-lists of P_x appears in the utility-list of P_y. For each tuple not appearing in P_y, the variable maxSupport is decremented by 1. If *maxSupport* is smaller than *lowerBound*(P_{xy}), the construction of the utility-list of P_{xy} can be stopped because the support of P_{xy} will not be higher than *lowerBound*(P_{xy}), and thus P_{xy} is not a correlated high utility itemset by Property 8, and its extensions can be ignored by Property 6. This pruning strategy is very effective and reduces execution time and memory usage by stopping utility-list construction early. Another similar pruning strategy called the LA-prune strategy, based on the utility of itemset P_{xy} instead of the bond measure, is also integrated into FCHM. [40].

Strategy 5. (LA-Prune)
Given two itemsets X and Y, if:

$$\sum_{\forall T_i \epsilon D} U(X,T_i) + RU(X,T_i) - \sum_{\forall T_j \epsilon D, X \subseteq T_j \text{ and } Y \nsubseteq T_j} U(X,T_j) + RU(X,T_j) < minutil$$

then $\forall Y' \supseteq Y$ *and* $X' \supseteq X$, $X'Y' \notin HUI$,.

The lemma condition consists of two parts. The first part calculates the total utility value of itemset *X* from transactions where *X* is present, regardless of the presence of *Y*. This includes all utility values associated with *X*. The second part subtracts the utility values of itemset *X* in transactions where *X* is present but *Y* is not. The sum of these two parts provides a more accurate upper bound on the utility of transactions where *XY* is likely to be present. If this resulting upper bound is lower than the *minutil*, then neither itemset *XY* nor any of its descendants can be considered as a *HUI*. Additionally, the utility value calculations are specific to itemset *X*, so none of its descendants that contain *Y* can be classified as *HUIs* either.

Strategy 6. Pruning Utility-List by upper-bound (PUL).
$FCHM_{bond}$ can generate a utility-list and a bit vector for each itemset P_{xy}. The bit vector can be obtained by scanning the database (for single items) or by performing the OR operation with two bit vectors. The bit vector of an itemset P_{xy} can then be used to determine $dissup(P_{xy})$. The construction of the utility-list of P_{xy} is skipped if the condition

$$\frac{Max\{support(P_x), support(P_y), support(xy)\}}{dissup(P_{xy})} < min\_measure$$

is satisfied. This is because the left-hand side of the inequality serves as an upper-bound on *bond*(P_{xy}).

2.3 Cosine measure

Cosine measure is a measure used to calculate correlation between items. The formula of cosine measure is defined as in [44] and [2].

- In the case we have two items A_1 and A_2, the cosine measure of these two items is calculated as:

$$cosine(A_1, A_2) = \frac{P(A_1 \cup A_2)}{\sqrt{P(A_1) \times P(A_2)}} = \frac{\sup(A_1 \cup A_2)}{\sqrt{\sup(A_1) \times \sup(A_2)}}$$

$cosine(A_1, A_2)$ calculates the probability of A_1 and A_2 co-occurring and divides it by the square root of the product of the probabilities of A_1 and A_2 individually. The square root ensures that the cosine value is solely determined by the support values of A_1, A_2, and their intersection $A_1 \cup A_2$,without being influenced by the total number of transactions. This characteristic makes the cosine measure *null-invariant*, as it remains unaffected by the presence of null transactions. In large datasets, where null transactions are common, the cosine-type measure delivers improved outcomes due to its ability to handle such scenarios effectively

- In the case we have more than two items, the cosine measure is extended as:

$$cosine(A_1, A_2, \dots, A_n) = \frac{P(A_1 \cup A_2 \cup \dots \cup A_n)}{\sqrt{P(A_1) \times P(A_2) \times \dots \times P(A_n)}} = \frac{sup(A_1 \cup A_2 \cup \dots \cup A_n)}{\sqrt{sup(A_1) \times sup(A_2) \times \dots \times sup(A_n)}}$$

2.4 Proposed approach

This thesis proposes another version of the FCHM algorithm named FCHM$_{cosine}$, which uses the cosine measure as the correlation measure for the FCHM algorithm. The two main reasons the cosine measure is preferred in this thesis are:

- Cosine measure is a null-invariant measure. [2,45]
- Cosine measure has the anti-monotonicity property (1)

2.4.1 Proof for anti-monotonicity property

From definition, we have

$$cosine(A_1, A_2, \dots, A_n) = \frac{sup(A_1 \cup A_2 \cup \dots \cup A_n)}{\sqrt{sup(A_1) \times sup(A_2) \times \dots \times sup(A_n)}} \quad (*)$$

$$cosine(A_1, A_2, \dots, A_n, A_{n+1}) = \frac{sup(A_1 \cup A_2 \cup \dots \cup A_n \cup A_{n+1})}{\sqrt{sup(A_1) \times sup(A_2) \times \dots \times \sup(A_n) \times \sup(A_{n+1})}} \quad (**)$$

Since $sup(A_1 \cup A_2 \cup \dots \cup A_n) \geq sup(A_1 \cup A_2 \cup \dots \cup A_n \cup A_{n+1})$ and $\sqrt{sup(A_1) \times sup(A_2) \times \dots \times sup(A_n)} \leq \sqrt{sup(A_1) \times sup(A_2) \times \dots \times \sup(A_n) \times \sup(A_{n+1})}$ then we can conclude that (*) ≥ (**). (2)

Denote *minimum cosine threshold* as α, from (2) we have:

$$cosine(A_1, A_2, \dots, A_n) < \alpha \Rightarrow cosine(A_1, A_2, \dots, A_n, A_{n+1}) < \alpha$$

This means if the itemset does not satisfy *minimum cosine* α, it is no need to traverse its superset.

2.4.2 Calculation of cosine measure

From the definition, the calculation of the *cosine(A)* value of an itemset *A* is depend on the two factors:

- product of support value of all 1-items in itemset *A*:

$$sup(A_1) \times sup(A_2) \times \dots \times \sup(A_n)$$

- the support value of itemset *A:*

$$sup(A_1 \cup A_2 \cup \dots \cup A_n)$$

To optimize performance, the product is calculated during the construction of the utility lists in FCHM algorithm. Specifically, in the Construct procedure , when the utility-list of itemset *Pxy* is constructed from utility-list of *Px* and utility-list of *Py*, this product can be calculated as

$product(Pxy) = product(Px) \times product(Py)$ if prefix *P* is null

else $product(Pxy) = \frac{product(Px) \times product(Py)}{product(P)}$.

Since support value of itemset *X* can be derived easily from utility list, the *cosine(X)* can be obtained efficiently.

These two following strategies from previous version of FCHM are also applied to further improve the performance of the FCHM$_{cosine}$ algorithm: DOS and PSN.

Chapter 3. Experiment and discussion

In this thesis, three benchmark datasets in the field of high utility itemset mining, namely *foodmart*, *mushroom*, and *retail*, are utilized to evaluate the experimental results of the proposed algorithm. Each dataset exhibits different characteristics that represent various types of data that algorithms may encounter in real life, thereby providing a more comprehensive assessment of the results. The general comparison between these three data sets is shown in Table 3.1. Also, the details of these three datasets are presented below.

Table 3.1. Dataset's characteristic

Dataset	**No. of distinct items**	**No. of transactions**	**Average transaction length**	**Type**	**Has real utility value?**
Foodmart	21,556	1,559	4.4	sparse with short transactions	Yes
Mushroom	88,162	16,470	23	dense	No
Retail	88,162	16,470	10.3	sparse with many items	No

3.1 Data

3.1.1 Mushroom dataset

The mushroom dataset consists of descriptions of hypothetical samples that represent 23 species of gilled mushrooms belonging to the Agaricus and Lepiota Family. Each species is categorized as either definitely edible, definitely poisonous, or of unknown edibility and not recommended. The dataset combines the latter class with the poisonous category. It is important to note that the accompanying Guide explicitly states that there is no straightforward rule for determining the edibility of a mushroom.

This is a benchmark dataset that has been widely used in the field of frequent itemset mining. In order to further leverage this real-life dataset for advanced tasks such as high utility itemset mining, the internal utility values have been generated using a uniform distribution ranging from 1 to 10. The original version of the dataset can be easily downloaded from the UCI Machine Learning Repository [41] or the Frequent Itemset Mining Dataset Repository [42], while the version with synthetic utility values can be obtained from the SPMF open-source data mining library [43].

3.1.2 Retail dataset

The retail dataset consists of market basket data obtained from an undisclosed Belgian retail supermarket. The data were collected during three separate time periods that were not consecutive. The first period spans from mid-December 1999 to mid-January 2000. The second period covers the duration from 2000 to early June 2000. The third and final period extends from late August 2000 to the end of November 2000. Regrettably, there is no available data between these periods. Consequently, the dataset encompasses approximately 5 months of data, with a total of 88,162 collected receipts.

Every entry in the dataset provides details such as the purchase date (recorded as 'date'), the receipt number (recorded as 'receipt nr'), the article number (recorded as 'article nr'), the quantity of items purchased (recorded as 'amount'), the article price in Belgian Francs (recorded as 'price' with 1 Euro equivalent to 40.3399 BEF), and the customer number (recorded as

'customer nr'). It's important to note that the article price in the dataset represents the unit price of the article multiplied by the quantity of items purchased.

Throughout the entire duration of data collection, the retail supermarket store stocks a total of 16,470 distinct Stock Keeping Units (SKU's). However, certain SKU's are only available seasonally, such as Christmas items. While the majority of products are identified by a unique barcode, some article numbers in the dataset represent a category or group of products rather than a specific individual item. This is evident with items like fruits, vegetables, meat, and a few others. During the data collection period, a total of 5,133 customers made at least one purchase at the supermarket.

Similar to the mushroom dataset, this is a real-life benchmark dataset in the field of frequent itemset mining, and the internal utility values have also been generated using a uniform distribution in the range from 1 to 10 to facilitate research in the area of high utility itemset mining. The original version of the retail dataset can be downloaded from the Frequent Itemset Mining Dataset Repository [42], and a version with synthetic utility values can be obtained from the SPMF open-source data mining library [43].

3.1.3 Foodmart dataset

The foodmart dataset comprises customer transaction data obtained from a retail store, extracted and converted from SQL-Server 2000. This dataset is also regarded as one of the benchmark datasets in the field of both frequent itemset mining and high utility itemset mining. The distinctive feature of this dataset compared to the Mushroom dataset and the retail dataset is that it contains real utility values. This means that synthetic utility values are not needed, making the problem addressed by the algorithm more realistic. This dataset can be downloaded from the SPMF open-source data mining library [43].

3.2 Analyze

The algorithms and all experiments are carried out in the environment with the following configuration: Intel(R) Core™ I3, 2.40GHz, memory capacity: 4 GB, operating system: Microsoft Windows 10, programming language: Java. The experiments used three evaluation criteria including effectiveness, runtime and memory consumption. The FCHM_{cosine} algorithm is in turn compared with the traditional HUIM algorithm which is FHM and some CHIM algorithms including FCHM_{bond} and $\text{FCHM}_{all-confidence}$. Several experiments are performed under various parameter. Specifically, paremeters are set up by fixing *minUtil* varying *minCore* and fixing *minCore* varying *minUtil*.

3.2.1 Effective Analysis

Table 3.2 compares the number of patterns between FCHM_{cosine} algorithm and FHM algorithm. FCHM_{cosine} with *minimum correlation* α is denoted as C_{α}. The various *minimum utility* value for each dataset is represented by the parameters from a_1 to a_5. In addition, the comparison between patterns count of the FCHM_{cosine}, FCHM_{bond} and $\text{FCHM}_{all-confidence}$ is also performed in Figure 3.1 and Figure 3.2.

Table 3.2. Compare patterns count with FHM

Dataset	**Algorithm**	**Number of patterns**				
		a_1	a_2	a_3	a_4	a_5
foodmart	**FHM**	233,231	231,904	219,012	154,670	59,351
	$\mathbf{C_{0.01}}$	101,629	100,303	87,966	36,252	3,274
	$\mathbf{C_{0.02}}$	81,511	80,222	68,745	25,409	2,530
	$\mathbf{C_{0.03}}$	48,912	47,687	3,7667	10,546	2,063
	$\mathbf{C_{0.04}}$	41,674	40,457	30,759	7,262	1,847
	$\mathbf{C_{0.1}}$	9,659	9,453	7,804	3,486	1,676
mushroom	**FHM**	1,045,780	585,013	273,448	179,215	92,656
	$\mathbf{C_{0.005}}$	1740	1379	921	711	435
	$\mathbf{C_{0.008}}$	501	406	303	253	178
	$\mathbf{C_{0.01}}$	207	140	85	59	37
	$\mathbf{C_{0.1}}$	161	109	63	40	20
	$\mathbf{C_{0.4}}$	160	109	63	40	20
retail	**FHM**	14,045	13,017	12,103	11,234	10,479
	$\mathbf{C_{0.1}}$	1910	1820	1741	1651	1575
	$\mathbf{C_{0.12}}$	1852	1765	1687	1598	1523
	$\mathbf{C_{0.14}}$	1812	1728	1650	1562	1488
	$\mathbf{C_{0.16}}$	1779	1696	1619	1533	1461
	$\mathbf{C_{0.4}}$	1,490	1,482	1,470	1,455	1,445

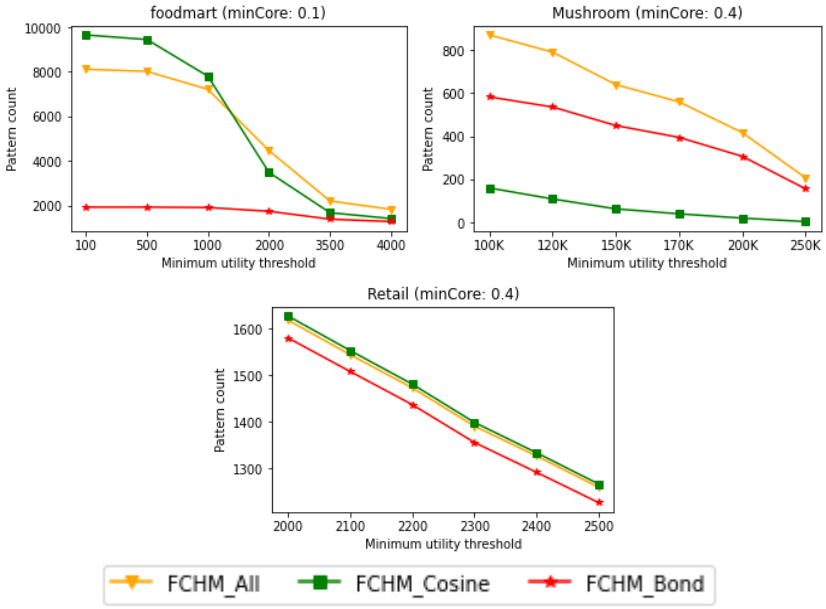

Figure 3.1. Compare pattern count with other versions (varying *minUtil*, fixing *minCore*)

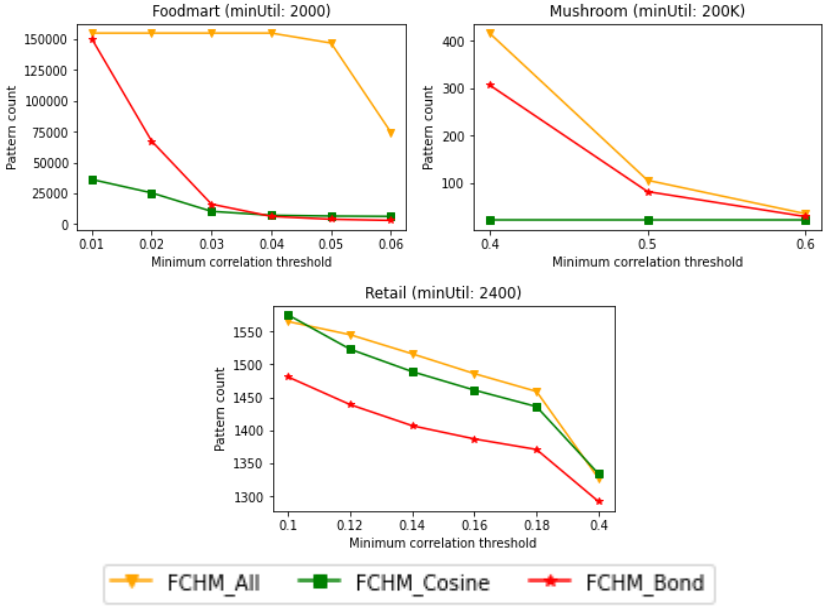

Figure 3.2. Compare pattern count with other versions (varying *minCore*, fixing *minUtil*)

From Table 3.2, it can be seen that the proposed algorithm helps to reduce a large number of weakly correlated patterns compared to the traditional HUIM algorithm FHM. Besides, Figure 3.1 and Figure 3.2 show that the number of patterns generated by the proposed algorithm is generally quite similar to the previous two versions of the FCHM algorithm. Except for the mushroom dataset, FCHM$_{cosine}$ returns significantly less patterns at small *minUtil* and *minCore*. This shows that the constraint set by the proposed algorithm can be considered tighter than previous versions in some cases.

3.2.2 Efficiency Analysis

The runtime of FCHM_{cosine} is compared with FHM (Figure 3.3, Figure 3.4) and FCHM_{bond}, $\text{FCHM}_{all-confidence}$ (Figure 3.5, Figure 3.6). Parameter values from Figure 3.1 and Figure 3.2 are preserved to ensure a fair comparison. It can be noticed that the runtime of FCHM_{cosine} is much improved compared to FHM. Specifically, in case the dataset contains many weakly correlated patterns like mushroom, the runtime of FCHM_{cosine} is 20 times faster than FHM. Besides, the runtime of the proposed algorithm is quite similar to $\text{FCHM}_{all-confidence}$. The reason is that although using different measures, the factors used to calculate these two measures are the same. Therefore, similar to $\text{FCHM}_{all-confidence}$, FCHM_{cosine} is also slower than FCHM_{bond} in the mushroom dataset because FCHM_{bond} has better prune search space than these two algorithms. However, in the remaining datasets, the use of bit vectors makes FCHM_{bond} slower than $\text{FCHM}_{all-confidence}$ and FCHM_{cosine}.

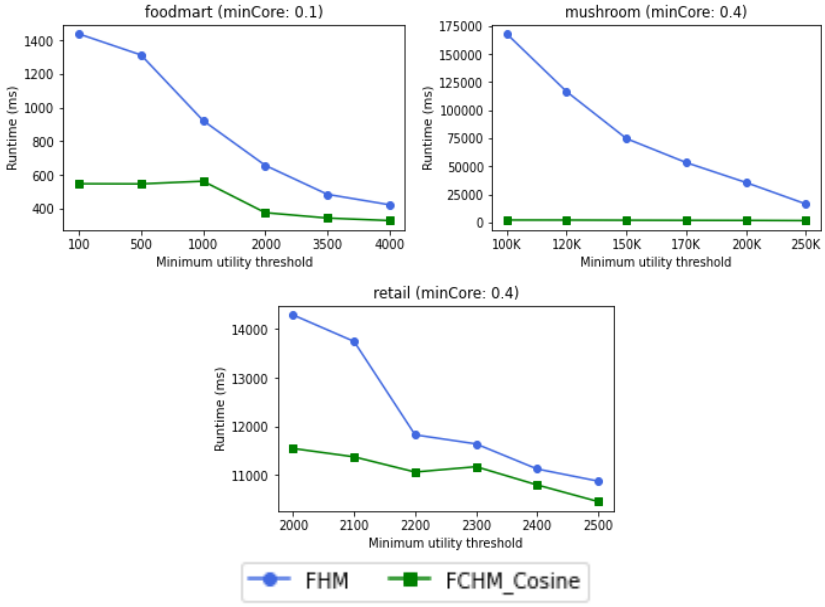

Figure 3.3. Compare runtime with FHM (varying *minUtil*, fixing *minCore*)

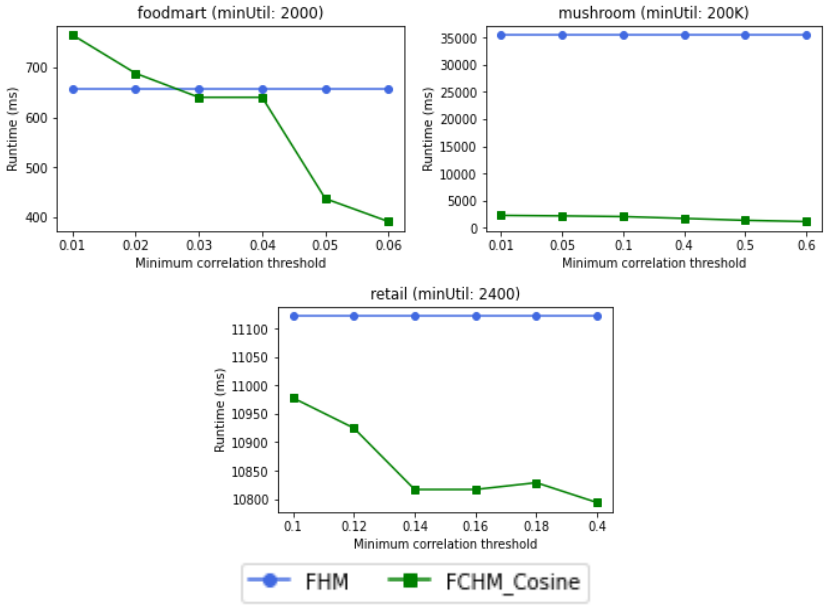

Figure 3.4. Compare runtime with FHM (varying *minCore*, fixing *minUtil*)

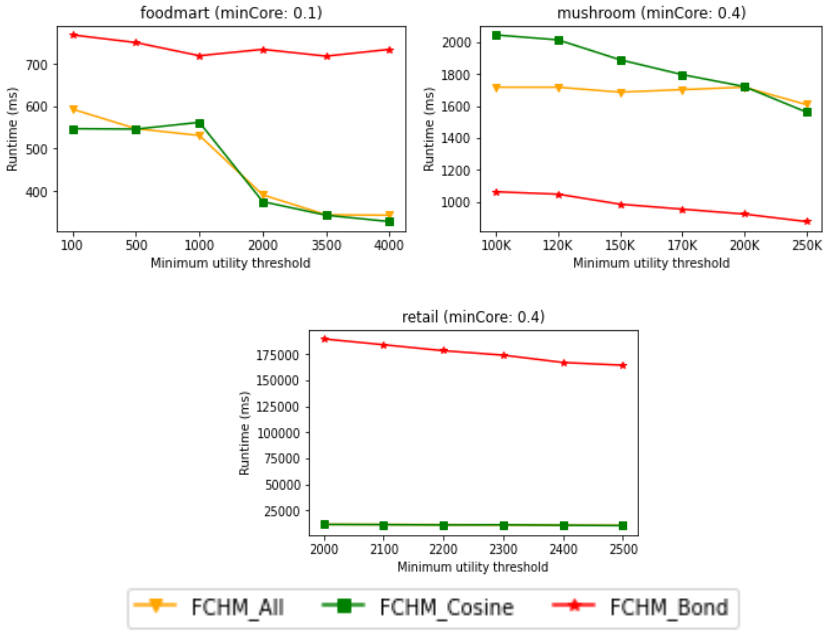

Figure 3.5. Compare runtime with other versions (varying *minUtil*, fixing *minCore*)

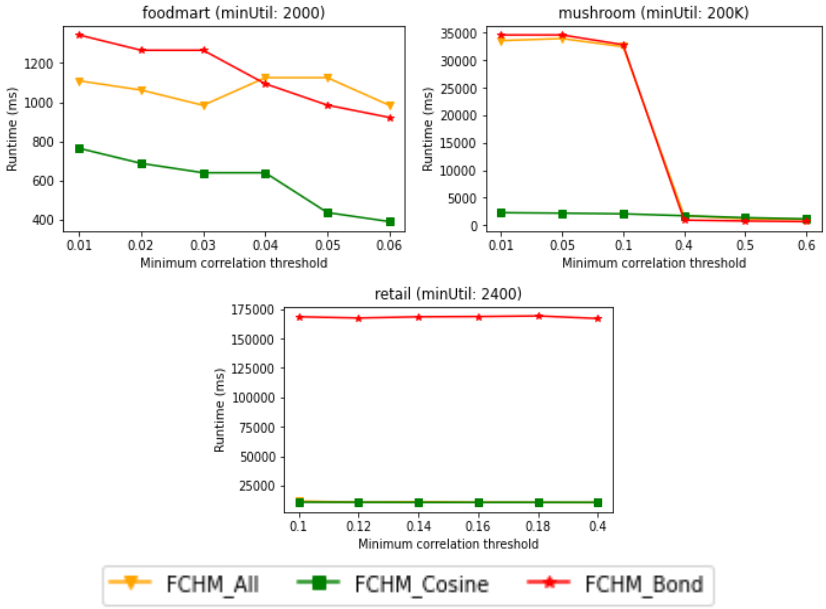

Figure 3.6. Compare runtime with other versions (varying *minCore*, fixing *minUtil*)

3.2.3 Memory Analysis

This section performs a memory evaluation between the compared algorithms above. Parameter values continue to be preserved. The results are shown in Figure 3.7 and Figure 3.8. In general, FCHM$_{all-confidence}$ and FCHM$_{cosine}$ have lower memory consumption than FHM algorithms because they can avoid a huge number of weakly correlated patterns and their measures can also be efficiently calculated. Specifically, the FCHM$_{cosine}$ algorithm is always in the top two algorithms with the lowest memory consumption. Besides, as mentioned above, the use of a disjunctive bit vector structure often causes FCHM$_{bond}$ to consume more memory than the other two versions of FCHM and even more than FHM in some cases. However, with the appropriate type of dataset and threshold, FCHM$_{bond}$ can still be the algorithm has the best memory consumption and runtime, the result on mushroom dataset is an example.

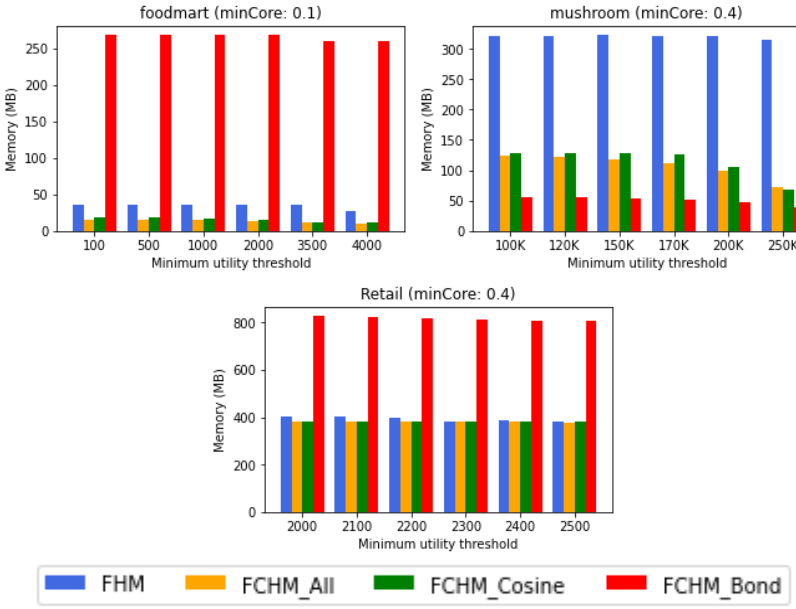

Figure 3.7. Compare memory with FHM and other versions (varying *minUtil*, fixing *minCore*)

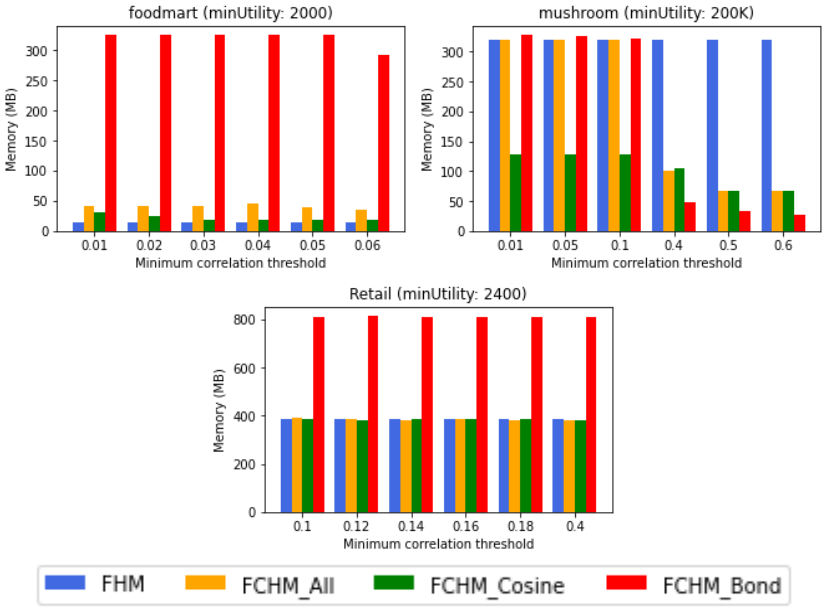

Figure 3.8. Compare memory with FHM and other versions (varying *minCore*, fixing *minUtil*)

3.3 Conclusion and Perspectives

This thesis proposes the FCHM$_{cosine}$ algorithm, which is a new version of the FCHM algorithm. The null-invariant and anti-monotonicity properties of the cosine measure are utilized to optimize the effectiveness and efficiency of the algorithm. Two strategies DOS and PSN from previous versions of the FCHM algorithm are also applied to effectively support the process of prune search space.

Experimental results show that the FCHM$_{cosine}$ algorithm significantly reduces weakly correlated patterns compared with the traditional HUIM algorithm. Besides, in general, the proposed algorithm has a stable runtime with memory consumption and in some cases better than the previous two versions of the FCHM algorithm.

In future study, the performance of the FCHM$_{cosine}$ algorithm can be further improved by developing some new pruning strategies that are suitable for the cosine measure. In addition, the study of some other measures, especially the null-invariant measures, may also produce interesting results.

References

1. Aggarwal, C.C.: Data Mining: The Textbook. Springer, Heidelberg (2015)
2. Han, J., Kamber, M., Pei, J.: Data mining concepts and techniques third edition, The Morgan Kaufmann Series in Data Management Systems (2011)
3. Fournier-Viger, P., Lin, J.C.-W., Kiran, R.U., Koh, Y.S., Thomas, R.: A survey of sequential pattern mining. Data Sci. Pattern Recognit. 1(1), 54–77 (2017)
4. Fournier-Viger, P., Lin, J.C.-W., Vo, B, Chi, T.T., Zhang, J., Le, H. B.: A survey of itemset mining. WIREs Data Mining and Knowledge Discovery, pp. e1207 (2017)
5. Agrawal, R., Srikant, R.: Fast algorithms for mining association rules. In: Proceedings of the 20th International Conference Very Large Data Bases, pp. 487–499. Morgan Kaufmann (1994)
6. Brauckhoff, D., Dimitropoulos, X., Wagner, A., Salamatian, K.: Anomaly extraction in backbone networks using association rules. IEEE/ACM Trans. Netw. 20(6), 1788–1799 (2012)
7. Duan, Y., Fu, X., Luo, B., Wang, Z., Shi, J., Du, X.: Detective: automatically identify and analyze malware processes in forensic scenarios via DLLs. In: Proceedings of the 2015 IEEE International Conference on Communications, pp. 5691–5696. IEEE (2015)
8. Fernando, B., Elisa F., Tinne T.: Effective use of frequent itemset mining for image classification. In: Proceedings of the 12th European Conference on Computer Vision, pp. 214–227. Springer (2012)
9. Glatz, E., Mavromatidis, S., Ager, B., Dimitropoulos, X.: Visualizing big network traffic data using frequent pattern mining and hypergraphs. Computing 96(1), 27–38 (2014)
10. Liu, Y., Zhao, Y., Chen, L., Pei, J., Han, J.: Mining frequent trajectory patterns for activity monitoring using radio frequency tag arrays. IEEE Trans. Parallel Distrib. Syst. 23(11), 2138– 2149 (2012)
11. Mukherjee, A., Liu, B., Glance, N.: Spotting fake reviewer groups in consumer reviews. In: Proceedings of the 21st International Conference on World Wide Web, pp. 191–200. ACM (2012)
12. Mwamikazi, E., Fournier-Viger, P., Moghrabi, C., Baudouin, R.: A dynamic questionnaire to further reduce questions in learning style assessment. In: Proceedings of the 10th International Conference Artificial Intelligence Applications and Innovations, pp. 224–235. Springer (2014)
13. Naulaerts, S., Meysman, P., Bittremieux, W., Vu, T.N., Berghe, W.V., Goethals, B., Laukens, K.: A primer to frequent itemset mining for bioinformatics. Brief. Bioinform. 16(2), 216–231 (2015)
14. Fournier-Viger, P., Wu, C.W., Zida, S., Tseng, V.S.: FHM: Faster high-utility itemset mining using estimated utility co-occurrence pruning. In: Proceedings of the 21st International Symposium Methodologies for Intelligent Systems, pp. 83–92. Springer (2014)
15. Lin, J.C.-W., Hong, T.P., Lu, W.H.: An effective tree structure for mining high utility itemsets. Expert Syst. Appl. 38(6), 7419–24 (2011)
16. Lin, Y.C., Wu, C.W., Tseng, V.S.: Mining high utility itemsets in big data. In: Proceedings of the Pacific-Asia Conference on Knowledge Discovery and Data Mining, pp. 649–661. Springer (2015)
17. Liu, M., Qu, J.: Mining high utility itemsets without candidate generation. In: Proceedings of the 21st ACM International Conference Information and knowledge management, pp. 55–64. ACM (2012)

18. Liu, Y., Liao, W.K., Choudhary, A.N.: A two-phase algorithm for fast discovery of high utility itemsets. In: Proceedings of the 9th Pacific-Asia Conference on Knowledge Discovery and Data Mining, pp. 689–695. Springer (2005)
19. Yao, H., Hamilton, H.J., Geng, L.: A unified framework for utility-based measures for mining itemsets. In: Proceeidngs of the ACM SIGKDD Workshop on Utility-Based Data Mining, pp. 28–37. ACM (2006)
20. Yun, U., Ryang, H., Ryu, K.H.: High utility itemset mining with techniques for reducing overestimated utilities and pruning candidates. Expert Syst. Appl. 41(8), 3861–3878 (2014)
21. Zida, S., Fournier-Viger, P., Lin, J.C.-W., Wu, C.W., Tseng, V.S.: EFIM: a highly efficient algorithm for high-utility itemset mining. In: Proceedings of the 14th Mexican International Conference Artificial Intelligence, pp. 530–546. Springer (2015)
22. Han, J., Pei, J., Ying, Y., Mao, R.: Mining frequent patterns without candidate generation: a frequent-pattern tree approach. Data Min. Knowl. Discov. 8(1), 53–87 (2004)
23. Zaki, M.J.: Scalable algorithms for association mining. IEEE Trans. Knowl. Data Eng. 12(3), 372–390 (2000)
24. Uno, T., Kiyomi, M., Arimura, H.: LCM ver. 2: efficient mining algorithms for frequent/closed/maximal itemsets. In: Proceedings of the ICDM'04 Workshop on Frequent Itemset Mining Implementations. CEUR (2004)
25. Pei, J., Han, J., Lu, H., Nishio, S., Tang, S., Yang, D.: H-mine: hyper-structure mining of frequent patterns in large databases. In: Proceedings of the 2001 IEEE International Conference Data Mining, pp. 441–448. IEEE (2001)
26. Tseng, V.S., Shie, B.-E., Wu, C.-W., Yu., P.S.: Efficient algorithms for mining high utility itemsets from transactional databases. IEEE Trans. Knowl. Data Eng. 25(8), 1772–1786 (2013)
27. Fournier-Viger, P., Lin, J.C.-W., Dinh, T., Le, H.B.: Mining correlated high-utility itemsets using the bond measure. In: Proceedings of the International Conference Hybrid Artificial Intelligence Systems, pp. 53–65. Springer (2016)
28. Omiecinski, E.R.: Alternative interest measures for mining associations in databases. IEEE Trans. Knowl. Data Eng. 15(1), 57–69 (2003)
29. Ahmed, C.F., Tanbeer, S.K., Jeong, B.S., Choi, H.J.: A framework for mining interesting high utility patterns with a strong frequency affinity. Inf. Sci. 181(21), 4878–4894 (2011)
30. Lin, J.C.-W., Gan, W., Fournier-Viger, P., Hong, T.-P., Chao, H.-C.: FDHUP: fast algorithm for mining discriminative high utility patterns. Knowl. Inf. Syst. 51(3), 873–909 (2016)
31. Bouasker, S., Ben Yahia, S.: Key correlation mining by simultaneous monotone and antimonotone constraints checking. In: Proceedings of the 30th Symposium on Applied Computing, pp. 851–856. ACM (2015)
32. Philippe, F.-V., Lin J.C.W., Dinh, T., Le, H.B.: Mining correlated high-utility itemsets using various measures. In: Logic Journal of the IGPL, Volume 28, Issue 1, Pages 19-32 (2020)
33. Gan, W., Lin, J.C.W., Philippe, F.-V., Chao, H.C., Fujita, H.: Extracting non-redundant correlated purchase behaviors by utility measure. In: Knowl.- Based Syst., vol. 143, pp. 30-41(2018)
34. Gan, W., Lin, J.C.W., Chao, H.C., Fujita, H., Yu, P.S.: Correlated utility-based pattern mining. In: Information Sciences, volume 504, pages 470-486, ISSN 0020-0255 (2019)
35. Vo, B. et al.: Mining Correlated High Utility Itemsets in One Phase. In: IEEE Access, vol. 8, pp. 90465-90477 (2020)
36. Sethi, K.K., Ramesh, D. (2021). Correlated High Average-Utility Itemset Mining. In: Bhateja, V., Peng, SL., Satapathy, S.C., Zhang, YD. (eds) Evolution in Computational

Intelligence. Advances in Intelligent Systems and Computing, vol 1176. Springer, Singapore. (2021)
37. Peng, A.X., Koh, Y.S., Riddle, P.: mHUIMiner: a fast high utility itemset mining algorithm for sparse datasets. In: Pacific-Asia Conference on Knowledge Discovery and Data Mining, pp. 196–207 (2017)
38. Duong, Q.H., Fournier-Viger, P., Ramampiaro, H., Norvag, K., Dam, T.-L.: Efficient high utility itemset mining using buffered utility-lists. Appl. Intell. 48(7), 1859–1877 (2017)
39. Qu, JF., Liu, M., Fournier-Viger, P.. Efficient Algorithms for High Utility Itemset Mining Without Candidate Generation. In: Fournier-Viger, P., Lin, JW., Nkambou, R., Vo, B., Tseng, V. (eds) High-Utility Pattern Mining. Studies in Big Data, vol 51. Springer, Cham. (2019)
40. S. Krishnamoorthy. Pruning strategies for mining high utility itemsets. Expert Systems with Applications, 42, 2371–2381. (2015)
41. UCI Machine Learning Repository, https://archive.ics.uci.edu/ml/datasets/mushroom. Accessed 21/5/2023
42. Frequent Itemset Mining Dataset Repository, http://fimi.uantwerpen.be/data/. Accessed 16/5/2022
43. Philippe, F.-V., Gomariz, A., Gueniche, T., Soltani, A., Wu, C., Tseng, V.S.: SPMF: a Java Open-Source Pattern Mining Library. In: Journal of Machine Learning Research, 15, 3389-3393 (2014)
44. Bagui, S., Just, J., Bagui, S.C., Hemashinha, R.: Using a cosine-type measure to derive strong association mining rules. In: Int. J. Knowl. Eng. Data Min. 1, 1, 69-83 (2010)
45. Tianyi, W., Yuguo, C., Jiawei, H.: Association Mining in Large Databases: A Re-Examination of Its Measures. In: Proceedings of the Int. Conf. Principles and Practice of Knowledge Discovery in Databases (PKDD'07) (2007)