FHUQI-Miner: Fast high utility quantitative itemset mining

Mourad Nouioua[1,2] · Philippe Fournier-Viger[1] · Cheng-Wei Wu[3] · Jerry Chun-Wei Lin[4] · Wensheng Gan[5]

Abstract

High utility itemset mining is a popular pattern mining task, which aims at revealing all sets of items that yield a high profit in a transaction database. Although this task is useful to understand customer behavior, an important limitation is that high utility itemsets do not provide information about the purchase quantities of items. Recently, some algorithms were designed to address this issue by finding quantitative high utility itemsets but they can have very long execution times due to the larger search space. This paper addresses this issue by proposing a novel efficient algorithm for high utility quantitative itemset mining, called FHUQI-Miner (Fast High Utility Quantitative Itemset Miner). It performs a depth-first search and adopts two novel search space reduction strategies, named Exact Q-items Co-occurrence Pruning Strategy (EQCPS) and Range Q-items Co-occurrence Pruning Strategy (RQCPS). Experimental results show that the proposed algorithm is much faster than the state-of-art HUQI-Miner algorithm on sparse datasets.

Keywords Pattern mining · quantitative pattern · High utility pattern · Quantities · Market basket analysis

1 Introduction

Data mining techniques can be generally described as descriptive or predictive [14]. The former are used to perform predictions, while the latter can summarize data or reveal interesting information from it to help users to understand the data. One of the main types of descriptive data mining is pattern mining, which aims at revealing interesting, useful or unexpected patterns in databases. Many pattern mining algorithms have been designed to find various types of patterns such as frequent itemsets [25], association rules [36] and frequent sequential patterns [13]. Early studies on pattern mining have mainly focused on finding frequent patterns, with the assumption that frequent patterns are interesting to users.

In recent years, motivated by the need to analyze more complex data and find more useful patterns, high utility pattern mining has emerged as a key pattern mining task. The goal is to find patterns that have a high importance as measured by a numeric utility function [8]. The utility can be used to measure the occurrence frequency but also other more interesting criteria. For example, to study purchasing habits in a customer transaction database, the utility of a pattern (a set of items) can be measured in terms of profit that it yields, while for analyzing click-stream data, utility can represent the time spent on webpages. Various types of high utility patterns have been studied such as high utility itemsets (HUIs) [8, 12, 35], high utility sequential patterns [1, 28, 31], high utility periodic patterns [4, 7] and high utility episodes [11, 21, 32]. Among these different kinds of patterns, *High Utility Itemset Mining* (HUIM) is the most studied problem.

✉ Philippe Fournier-Viger
philfv8@yahoo.com

Mourad Nouioua
mouradnouioua@gmail.com

Cheng-Wei Wu
wucw@niu.edu.tw

Jerry Chun-Wei Lin
jerrylin@ieee.org

Wensheng Gan
wsgan001@gmail.com

1 School of Humanities and Social Sciences, Harbin Institute of Technology (Shenzhen), Shenzhen, China

2 University of Bordj Bou Arreridj, Bordj Bou Arreridj, Algeria

3 Department of Computer Science and Information Engineering, National Ilan University, Yilan City, Taiwan

4 Department of Computing, Mathematics and Physics, Western Norway University of Applied Sciences (HVL), Bergen, Norway

5 College of Information Science and Technology, Jinan University, Guangzhou, China

The goal of HUIM is to enumerate all sets of items that have a utility that is no less than a user-defined minimum utility threshold (e.g. all sets of products purchased together that yield a high profit). The input database format of HUIM is a database of transactions with weight and quantities. This format can be used to model data from many domains and it is richer than the format used for the traditional problem of *Frequent Itemset Mining* (FIM) [9, 25]. More precisely, the input database in FIM is a table where each transaction (row) is described using binary attributes (items) such that each item can appear no more than once in each transaction, and all items are considered as equally important. In HUIM, these restrictions are lifted. Each item can appear more than once in each transaction, and each item can have a utility value (a weight) indicating its relative importance in terms of factors such as profit, cost, and time [8, 12, 17, 35].

HUIM has numerous applications. However, a critical limitation of traditional HUIM algorithms is that the discovered patterns do not provide information about quantities to the user, even though quantities are encoded in the input database. To address this limitation, *High Utility Quantitative Itemset Mining* (HUQIM) was proposed as an extension of HUIM. The objective of HUQIM is to discover all sets of items that have a high utility while also providing information about item quantities that led to this utility [18, 19, 30, 33]. The additional information about item quantities can be very useful as illustrated by the following example. Consider the analysis of a costumer transaction database. Applying traditional HUIM algorithms on this data will identify sets of items (itemsets) such as *"coffee, cookies"* that yield a high profit. This information is interesting for purposes such as marketing as these items could be co-promoted to increase sales. However, this kind of patterns does not inform the user about how many boxes of coffees or how many cookies a customer typically buy. HUQIM algorithms addresses this issue by discovering high utility itemsets that also indicate quantities, which is more accurate and thus more useful for taking decisions and understanding customer behaviors [33]. For instance, a HUQIM algorithm could discover a quantitative high utility itemset *"coffee:3, cookies:2, eggs:6"*, indicating that buying 3 boxes of coffee with 2 cookies, and 6 eggs yields a high profit. Note that HUQIM is not restricted to finding patterns with single quantities. It can also find patterns containing range of quantities. For example, a pattern *"cheese:3-6, juice:5-7"* indicates that buying 3 to 6 pieces of cheese with 5 to 7 bottles of juice generates a high profit. It can be seen from this example, that patterns found by HUQIM are more informative than those found by HUIM since quantity information is provided. This information can help decision makers to take more accurate decisions. For example, by designing tailored promotions such as offering a discount to customers that buy at least 7 bottles of juice with 5 pieces of cheese.

Hence, HUQIM can be viewed as more useful than HUIM. However, HUQIM is much more difficult than HUIM. The reason is that while HUIM does not consider item quantities in patterns, HUQIM associates a quantity or range of quantities to each item. Then, the same item with two different quantities can be viewed as distinct quantitative items (Q-items). As a result, the search space of HUQIM is much larger than that of HUIM. An itemset that is composed of a set of quantitative items is called a *quantitative itemset (Q-itemset)*. A *Q-itemset* is said to be a *high utility quantitative itemset* if its utility is no less than *a user-specified minimum utility threshold* θ.

Despite the proposal of several HUQIM algorithms [18, 19], a major issue is that they can still have very long runtimes due to the very large search space. This is inconvenient for users who must often wait a long time to obtain results. Hence, it is desirable to propose more efficient algorithms based on novel search space pruning strategies to efficiently reduce the search space and therefore make the task of HUQIM faster.

This paper addresses this issue by proposing a novel algorithm called FHUQI-Miner (*Fast High Utility Quantitative Itemset Miner*). It is based on two novel search space pruning strategies, namely the *Exact Q-items Co-occurrence Pruning Strategy* (EQCPS) and *Range Q-items Co-occurrence Pruning Strategy* (RQCPS). These strategies allows to eliminate unpromising itemsets early from the search space. This paper also describes an extensive experimental evaluation to compare the performance of FHUQI-Miner with the previous state-of-the-art HUQI-Miner algorithm. Results show that FHUQI-Miner outperforms HUQI-Miner on sparse datasets.

The rest of this paper is organized as follows: The next section presents a review of the main approaches for mining high utility itemsets and high utility quantitative itemsets. Section 3 introduces the background related to HUQIM. Section 4 describes the utility-list structure and how it can be used in HUQIM. Section 5 presents the proposed FHUQI-Miner algorithm and its proposed pruning strategies. Section 6 presents the experimental evaluation of FHUQI-Miner. Finally, a conclusion is drawn and some research opportunities are discussed in Section 7.

2 Related work

FIM [9, 25] is a fundamental data mining task, which aims at finding all sets of items that appear at least some minimum number of times in a transaction database. Finding frequent

itemsets is useful but is based on the assumption that frequent patterns are interesting. For some applications, other importance criteria are more appropriate than the frequency such as the amount of profit obtained by the sales of item. Besides, another limitation of FIM is that items have binary quantities (presence or absence of each item) in transactions and all items are treated as equally important. Several FIM algorithms have been proposed [2, 9, 15]. They take advantage of the anti-monotonicity of the support (occurrence frequency) to reduce the search space. This property states that an itemset cannot have a support greater than that of its subsets. Hence, all supersets of an infrequent itemset do not need to be considered, which speeds up the discovery of frequent itemsets.

FIM was extended as HUIM to address its aforementioned limitations [3, 23, 24, 29]. In HUIM, each item can have non binary quantities in transactions (e.g. a customer may purchase five breads and three apples), called internal utility. Moreover, each item can have a numeric value, called external utility, indicating its relative importance (e.g. selling one apple yield a 1 $ profit while selling a bread yield a 2$ profit). The goal of HUIM is then to find all sets of items that have a utility that is not less than some minimum utility threshold specified by the user. In HUIM, the utility is a numeric function that assesses the importance or value of a set of items. For applications such as market basket analysis, it can represent the amount of profit generated by the sale of items, but HUIM can also be applied in other contexts such as web click-stream analytics where utility could measure other aspects such as the amount of time spent on a website. HUIM can reveal interesting insights in real data such as itemsets that yield a high profit but are infrequent, which are ignored by FIM. However, finding high utility itemsets is more difficult than mining frequent itemsets because the utility function typically considered in HUIM is neither monotonic nor anti-monotonic. In other words, the utility of an itemset may be greater than that of its subsets or supersets. For that reason, strategies used to efficiently mine frequent itemsets are not directly applicable to solve the HUIM problem.

In recent years, several HUIM algorithms were proposed. These algorithms can be roughly classified in two categories: Two-phase based approaches and one-phase based approaches. As indicated in its name, two-phase based approaches discover HUIs in two phases. To reduce the search space, they rely on upper bounds on the utility that are anti-monotonic such as the *Transaction-Weighted Utility* (TWU) upper bound [24]. In the first phase, a set of candidate HUIs are generated by overestimating the utility of itemsets, and thus these candidates may contain low utility itemsets (LUIs). However, the first phase does not underestimate the utility of high utility itemsets. Hence, some LUIs are selected in the set of candidates during the first phase. Then, the second phase consists of calculating the exact utility of candidate itemsets to filter out LUIs. The first two-phase based algorithm, abbreviated as TP, was proposed by Liu et al. [24]. Then, other more efficient two-phase-based algorithms were designed such as IHUP [20], UP-Growth [29] and HUP-Growth [34].

A fundamental problem of two-phase based algorithms is the generation of a very large number of candidate itemsets in the first phase, especially for large databases or databases with long transactions. Consequently, the cost of scanning the database to calculate the exact utility of each candidate itemset in the second phase can also be very high. Hence, two-phase based algorithms may have long runtimes and may consume much memory.

To address this problem, one-phase based algorithms were introduced. They immediately identify low and high utility itemsets in only one phase without generating candidate itemsets. Moreover, another improvement of one phase based algorithms is that they use tighter upper bounds on the utility of itemsets that are based on the exact utilities of itemsets.

HUI-Miner is the first one-phase based algorithm [23]. It is based on a novel data-structure called *Utility-List* (UL). The utility-list of an itemset stores all necessary information to quickly calculate the itemset's utility without reading the database. Moreover, using Utility-Lists(UL_s), HUI-Miner can directly calculate an upper bound on the utility of an itemset and all its extensions to reduce the search space. This upper bound is based on a concept of remaining utility, representing the utility that could be used when extending an itemset with additional items. HUI-Miner starts by constructing utility-lists of itemsets having a single item (1-itemsets). Then, HUI-Miner recursively builds utility-lists of larger itemsets by joining utility-lists of their subsets. Despite the fact that, HUI-Miner was shown to outperform the state-of-the-art two-phase based algorithms, it was observed that the join operation used to obtain utility-lists of larger itemsets remains costly in terms of running time. To deal with this problem, another efficient algorithm called Faster High-Utility Itemset Mining Algorithm (FHM) has been proposed [10].

FHM was designed with the goal of reducing as much as possible the number of join operations. To this end, an additional pruning strategy, called *Estimated Utility Co-occurrence Pruning* (EUCP), is adopted that can prune some low utility itemsets without performing the join operation. The proposed strategy first calculates the TWU measure of all pairs of items that co-occur in the database. Then, during the recursive pattern mining process, low TWU itemsets (LTWUIs), i.e, itemsets having TWU values

that are less than the pre-defined minimum utility value θ, are eliminated early as well as all their supersets. It was shown that, this strategy can greatly reduce the search space as FHM was found to be much faster than HUI-Miner.

Using similar approaches, several other one phase based algorithms have been proposed such as: mHUIMiner [26], d^2HUP [22], EFIM [37], HMiner [16], HUI-Miner* [27] and ULB-Miner [5]. In those studies, it was observed that one-phase based algorithms generally outperform two-phase based algorithms. However, both types of algorithms only discover patterns that generate a high profit in databases without giving information to the user about item quantities. To overcome this limitation, HUQIM algorithms came into play where both utilities and quantities of itemsets are taken into account. The first algorithm for mining high utility quantitative itemsets is HUQA [33]. HUQA introduced the concept of *weak utility quantitative itemsets* to efficiently discover *High Utility Quantitative Itemsets* (HUQIs). Weak utility quantitative itemsets are itemsets that can be extended to get high utility quantitative itemsets. To prune unpromising Q-itemsets, the k-support bound measure is used.

Motivated by the success of utility-list based algorithms, a vertical based algorithm (VHUQI) for HUQIM was proposed [19]. It utilizes a variation of the utility-list structure. Experimental results have shown that VHUQI outperforms HUQA. However, a key problem with these two algorithms is that they only adopt the k-support bound to eliminate Low Utility Quantitative Itemsets (LUQIs) which is insufficient to reduce the search space.

To address this problem, Li et al. [18] have recently proposed a novel algorithm, named *HUQI-Miner (High Utility Quantitative Itemsets Miner)*, where both the TWU and a *remaining utility*-based upper bound are used to quickly eliminate LUQIs.

Although HUQI-Miner was found to outperform previous algorithms, the runtime of HUQI-Miner is still very long due to the huge number of join operations that are performed during the mining process. A reason is that the same item with two different quantities is regarded as two different Q-items in HUQIM. Accordingly, the search space in HUQIM is much larger than the search space in HUIM. As a result, the number of join operations for HUQIM can be much larger than in HUIM.

To overcome this limitation, this paper proposes a novel improved algorithm named FHUQI-Miner where the EUCP strategy [10] utilized in traditional HUIM is modified and extended to deal with Q-itemsets in HUQIM. More precisely, two new pruning strategies, EQCPS and RQCPS, are proposed and used for mining HUQIs.

3 Background

This section first describes preliminary concepts of HUQIM. Then, the combining operation of HUQIM is presented. After that, the formulation of the HUQIM problem is given. Finally, the TWU pruning strategy is introduced.

3.1 Preliminaries

Let $I = \{I_1, I_2, \ldots, I_N\}$ be a set of N *distinct items*, a *quantitative transaction database* D is composed of a set of transactions, denoted as $D = \{T_1, T_2, ..., T_M\}$, where each transaction $T_q \in D$ $(1 \leq q \leq M)$ has a unique identifier called T_{id} (Transaction Identifier) and each transaction is a subset of I. Besides, every item $i \in I$ that appears in a transaction T_q has a positive number $q(i, T_q)$, called *internal utility*, which represents the quantity of item i in T_q. Moreover, each item i has a profit p_i (a positive number) called *external utility*.

Table 1 presents an example of a quantitative transaction database D, which is composed of four transactions, T_1 to T_4. Moreover, Table 2 presents the external utilities of items in D. The database D will be used as example through this section to illustrate the different concepts of HUQIM. Taking item A as example, we can see in Table 2 that the profit of item A is 3, and in Table 1 that the internal utility of A in T_1 (resp. T_2, T_3, T_4) is 2 (resp. 0, 2, 2).

In HUQIM, there are two kinds of quantitative items: Exact quantitative items, also called *Exact Q-items*, and range quantitative items, also named *Range Q-items*.

An *exact Q-item* x is defined as a pair (i, q) where $i \in I$ and q is the quantity of item i. Thus, each transaction $T_q \in D$ is composed of a set of exact Q-items, $T_q = \{x_1, x_2, \ldots, x_k\}$.

For example, the transaction of T_1 in Table 1 is composed of exact Q-items (A,2), (B,5), (C,2) and (D,1).

A *range Q-item* is another type of Q-items where the quantity of the corresponding item has not a unique value but it is defined as a range. *Range Q-items* do not exist explicitly in the database but they are obtained by combining exact Q-items. A *range Q-item* x is defined as a triple (i, l, u) where $i \in I$, l (resp. u) represents the *lower* (resp. *upper*) *bound* of the quantity of item i. The interval size of a range Q-item, called a *Q-interval*, is defined as

Table 1 An example of a transaction database

T_{id}	Transaction
T_1	(A,2) (B,5) (C,2) (D,1)
T_2	(B,4) (C,3)
T_3	(A,2) (C,2)
T_4	(A,2) (B,6) (D,1)

Table 2 Profit table

Item	A	B	C	D
Profit	3	1	2	2

$(u - l + 1)$. Note that, an exact Q-item (i, q) can be formulated as a range Q-item (i, l, u) where $q = l = u$.

For example, (A,5,7) is a range Q-item that has a Q-interval of size 3.

A *quantitative itemset* X, denoted as *Q-itemset X*, is a set of Q-items. A *k-Q-itemset* is a Q-itemset consisting of k distinct Q-items, $X = [x_1, x_2, \ldots, x_k]$. If X is composed only of exact Q-items, X is an *exact Q-itemset*. If there is at least one range Q-item, X is a *range Q-itemset*.

For example, [(A,5), (B,6), (D,3)] is an exact Q-itemset. Or more precisely, an exact 3-Q-itemset. [(B,6), (D,3,5)] is a range Q-itemset. More precisely, it is a range 2-Q-itemset.

Definition 1 (Inclusion of Q-items) Given an exact Q-item $x = (i, q)$ and a range Q-item $y = (j, l, u)$, we say that y *includes* x, or x *is included in* y, if $i = j$ and $l \leq q \leq u$.

Given two range Q-items, $x = (i, l, u)$ and $y = (i', l', u')$, y *includes* x, or x *is included in* y, if $i = j, l \geq l'$ and $u \leq u'$.

For example, the exact Q-item (A,3) is included in the range Q-item (A,1,5).

Definition 2 (Occurrence of a Q-item) An exact Q-item $x = (i, q)$ *occurs in a transaction* $T_d = \{y_1, y_2, \ldots, y_k\}$ if $x \in T_d$.

A range Q-item $x = (i, l, u)$ *occurs in a transaction* $T_d = \{y_1, y_2, \ldots, y_k\}$ if one of its included exact Q-items occurs in this transaction. Formally, x *occurs in transaction* T_d if there exists a Q-item $y = (j, q') \in T_d$ such that $i = j$ and $l \leq q' \leq u$.

For example, (B,4) appears in T_2, while (B,4,6) appears in transactions T_1, T_2 and T_4.

Definition 3 (Occurrence of a Q-itemset) A Q-itemset $X = \{x_1, x_2, \ldots x_k\}$ *occurs in a transaction* T_d if $\forall x \in X$, x occurs in T_d.

For example, we can observe that Q-item (A,2) occurs in transactions T_1, T_3 and T_4 and that the Q-itemset [(A,2),(D,1)] occurs in transactions T_1 and T_4.

Definition 4 (Occurrence-set of a Q-itemset) The *occurrence-set* of a Q-itemset X, denoted as $OCC(X)$, it the set of transactions where X appears.

For example, $OCC([(A, 2), (C, 2)]) = \{T_1, T_3\}$.

Definition 5 (Support count of a Q-itemset) Given a Q-itemset X, the *support count* of X, denoted as $SC(X)$, is defined as the number of transactions where X appears, i.e., $SC(X) = |OCC(X)|$ [19].

For example, $SC([(A, 2), (C, 2)]) = |\{T_1, T_3\}| = 2$.

Definition 6 (Utility of a Q-item in a transaction) The *utility of an exact Q-item* $x = (i, q)$ in a transaction T_d, denoted as $u(x, T_d)$, is defined as $u(x, T_d) = p_i \times q$ [19].

The *utility of a range Q-item* $x = (i, l, u)$ in a transaction T_d is the sum of utilities of all exact Q-items that are included in x. Formally, $u(x, T_d) = \sum_{j=l}^{u} u((i, j), T_d)$ [19].

For example, $u((A, 2), T_1) = 3 \times 2 = 6$. Moreover, $u((A, 2, 3), T_3) = u((A, 2), T_3) + u((A, 3), T_3) = 6 + 0 = 6$

Definition 7 (Utility of a Q-itemset in a transaction / in the database) Given a Q-itemset $X = [x_1, x_2, \ldots, x_k]$, the *utility of a Q-itemset X in a transaction* T_d, denoted as $u(X, T_d)$, is the sum of utilities of Q-items from X in T_d. Formally, $u(X, T_d) = \sum_{j=1}^{k} u(x_j, T_d)$ [19].

The *utility of a Q-itemset X in a database D*, denoted as $u(X)$, is the sum of utilities of X in all transaction where X occurs. Formally, $u(X) = \sum_{T_d \in OCC(X)} u(X, T_d)$ [19].

For instance, $u([(A, 2, 3)(C, 2, 3)], T_1) = u((A, 2, 3), T_1) + u((C, 2, 3), T_1) = 6 + 4 = 10$. Similarly, $u([(A, 2)(C, 2)]) = u((A, 2)(C, 2), T_1) + u((A, 2)(C, 2), T_3) = 10 + 10 = 20$.

Definition 8 (Utility of a transaction) The *utility of a transaction* $T_d = \{y_1, y_2, \ldots, y_k\}$, denoted as $TU(T_d)$, is the sum of utilities of all Q-items that occurred in T_d, that is $TU(T_d) = \sum_{i=1}^{k} u(y_i, T_d)$ [19].

For example, $TU(T_2) = u((B, 4), T_2) + u((C, 3), T_2) = 4 + 6 = 10$

Definition 9 (Total utility of a database) The *total utility of a database D*, denoted as σ, is the sum of its transaction utilities. Formally, $\sigma = \sum_{T_d \in D} TU(T_d)$ [19].

For example, the total utility of the database D given in Table 1 is $\sigma = TU(T_1) + TU(T_2) + TU(T_3) + TU(T_4) = 17 + 10 + 10 + 14 = 51$.

Definition 10 (High utility quantitative itemset) Given a user-defined *minimum utility threshold* $\theta (0 \leq \theta \leq \sigma)$ and a Q-itemset X, X is a *high utility quantitative itemset*, abbreviated as *HUQI*, if the utility of X is no less than θ. Otherwise, X is a *low utility quantitative itemset*, abbreviated as *LUQI* [19].

For example, if the minimum utility threshold θ is set to 15, then the Q-itemset [(A,2), (C,2)] is a HUQI because $u([(A, 2), (C, 2)]) = 20 > 15$.

Similarly to HUIM [23], the utility function of HUQIM is also *not monotonic* nor *anti-monotonic*. In other words, the utility of a Q-itemset may be less, greater, or equal to the utility of its supersets. Accordingly, alternative pruning strategies are necessary to prune unpromising itemsets. Before presenting the TWU pruning strategy, we first introduce some concepts and definitions that are related to the combination process of Q-itemsets.

3.2 Q-itemsets combination

As mentioned above, range Q-itemsets do not explicitly exist in the database, but they are produced by combining exact Q-itemsets. In fact, mining range Q-itemsets in addition to exact Q-itemsets is more advantageous since it gives the opportunity to discover more interesting patterns by providing quantity range information within the discovered patterns.

The combination operation is performed by merging Q-itemsets that have Q-items with consecutive quantities. To apply the combination process, it is required to define: (1) The combining method to be applied.

(2) The set of candidate Q-itemsets to be possibly combined.

(3) *A quantitative related coefficient (qrc)* [33].

Three combining methods have been proposed for HUQIM, namely $Combine\_Max$, $Combine\_Min$ and $Combine\_All$ [6, 18]. In the next section, an explanation of each combining method is given with illustrative examples.

Candidate Q-itemsets are a set of LUQIs that may be used to perform the combination process. It is worth noticing that, not all LUQIs are selected to be candidate Q-itemsets but only those having utility values that are high enough. The selection of candidate Q-itemsets for combination is done based on the obervation that although some Q-itemsets are LUQIs, they have a utility values that are close to that required for HUQIs. Therefore, it is highly probable that some combinations of these Q-itemsets will produce high utility range Q-itemsets.

The qrc coefficient was introduced to avoid continuously combining Q-itemsets with adjacent quantities, which may produce range Q-itemsets with very large Q-intervals [33]. Such kind of range Q-itemsets are undesirable since their interpretation may be meaningless. Therefore, the combination process should be stopped if the Q-interval of a generated Q-itemset is larger than qrc. Moreover, qrc is also used to avoid combining LUQIs and considering only some Q-itemsets that may lead to produce HUQIs.

Formally, candidate Q-itemsets and the combining constraint are respectively defined as follows:

Definition 11 (The candidate quantitative itemset) Given *a user-defined minimum utility threshold* θ where $0 \leq \theta \leq \sigma$ and a *quantitative related coefficient* ($qrc > 0$), a *Q-itemset* X is a *candidate Q-itemset* if $\frac{\theta}{qrc} \leq u(X) \leq \theta$.

Definition 12 (Last Q-items combining constraint) Given a *quantitative related coefficient* ($qrc > 0$) and two *Q-itemsets* $X = [(x_1, l_1, u_1), (x_2, l_2, u_2), \ldots, (x_k, l_k, u_k)]$ and $Y = [(y_1, l'_1, u'_1), (y_2, l'_2, u'_2), \ldots, (y_k, l'_k, u'_k)]$.

X and Y *can be combined* together to form a range Q-itemset $Z = [(i_1, l_1, u_1), (i_2, l_2, u_2), \ldots, (i_k, l_k, u'_k)]$, if the following conditions hold [18]:

(1) X and Y are both candidate Q-itemsets.
(2) X and Y have the same prefix. i.e, the first $(k-1)$ Q-items in X and Y are the same. Formally, $\forall(1 \leq i \leq k-1)$: $x_i = y_j, l_i = l'_j$ and $u_i = u'_j$.
(3) For the last Q-item, $x_k = y_k$ and $l'_k = (u_k + 1)$.
(4) The Q-interval of the last Q-item to be generated should be less than qrc, i.e, $u'_k - l_k \leq qrc$.

It can be seen that, the combination process is always performed on the last Q-item of two Q-itemsets that have a same prefix and quantities of their last Q-items are consecutive.

For example, suppose that $qrc = 3$ and that there are 4 candidate Q-itemsets X_1=[(A,2), (B,3), (C,5)], X_2=[(A,2), (B,3), (C,6)], X_3=[(A,2), (B,3), (C,7)] and X_4=[(A,2), (B,3), (C,8)]. X_1 can be combined with X_2 because the four conditions of the combining constraint are verified (Definition 12). The result of combining X_1 and X_2 is Y_1=[(A,2), (B,3), (C,5,6)]. Again, Y_1 can be combined with X_3 to form Q-itemset Y_2=[(A,2), (B,3), (C,5,7)]. However, Y_2 cannot be combined with X_4 because the *Q-interval* of the last Q-item in the generated Q-itemset [(A,2), (B,3), (C,5,8)] is larger than qrc.

3.2.1 Combining methods

Based on the combining constraint presented in Definition 12, there exist three combining methods in HUQIM, namely $Combine\_Max$, $Combine\_Min$ and $Combine\_All$. These methods differ from each other on how the combining constraint is used [6, 18]. Basically, candidate Q-itemsets have the same prefix and they differ only on the last Q-item. Thus, Q-itemsets are first ordered according to the last Q-item. Then, the combining method continuously combines each Q-itemset with Q-itemsets that come after according to a processing order. This process is repeated until traversing all candidate Q-itemsets.

Combine_All method The $Combine\_All$ method outputs all possible high utility range Q-itemsets that can be generated by either combining candidate Q-itemsets or

by combining candidate Q-itemsets with range Q-itemsets [18].

To better illustrate the *Combine_All* method, let's assume that there is a sorted set C of 6 candidate Q-itemsets {[(A,2),(B,6)], [(A,2),(C,4)], [(A,2), (C,5)], [(A,2),(C,6)], [(A,2),(C,7)], [(A,2),(C,9)]} and $qrc = 4$. The combining process on C using the *Combine_All* method is illustrated in Fig. 1.

Besides, arrows are labeled with numbers that indicate the order of generation of range Q-itemsets. Moreover, colors are used to show range Q-itemsets that can be generated after traversing one candidate Q-itemset. The method first generates Q-itemsets in rectangles with rounded corners, followed by Q-itemsets in rectangles with sharp corners, and then Q-itemsets in Parallelograms.

The method tries to combine each Q-itemset in C with all Q-itemsets that come after this current Q-itemset. Back to our example, the method tries to combine the first Q-itemset in C which is [(A,2),(B,6)] with the second Q-itemset [(A,2),(C,4)]. However, since the 3^{rd} condition of the combining constraint (Definition 12) is not valid, the method passes directly to the second Q-itemset. The second Q-itemset [(A,2),(C,4)] can be combined with the third Q-itemset [(A,2),(C,5)] because all conditions of the combining constraint are verified. The resulting Q-itemset is [(A,2),(C,4,5)]. Similarly, this range q-itemsets is combined with the next Q-itemset in C which is [(A,2),(C,6)] and Q-itemset [(A,2),(C,4,6)] is generated. Once again, the method combines [(A,2),(C,4,6)] with [(A,2),(C,7)] to generate [(A,2),(C,4,7)]. Moving to the next Q-itemset, the method does not combine [(A,2),(C,4,7)] with [(A,2),(C,9)] because the 3^{rd} condition of the combining constraint presented in definition 12 is not satisfied. After traversing the Q-itemset [(A,2), (C,4)], three range Q-itemsets are generated which are [(A,2), (C,4,5)], [(A,2), (C,4,6)] and [(A,2), (C,4,7)].

At this point, the method passes to the next Q-itemset in C, [(A,2),(C,5)] and repeats the above process by trying to combine this Q-itemset with the rest of Q-itemsets that come after it. As a result, Q-itemsets [(A,2),(C,5,6)] and [(A,2),(C,5,7)] are generated. The above process is repeated with the rest of Q-itemsets until traversing all Q-itemsets.

It is worth noticing that, at each generation of a new range Q-itemset, the method checks its utility to keep only range Q-itemsets having high utilities. More precisely, the method will keep only HUQIs from the set of generated Q-itemsets {[(A,2),(C,4,5)], [(A,2),(C,4,6)], [(A,2), (C,4,7)], [(A,2),(C,5,6)], [(A,2),(C,5,7)], [(A,2),(C,6,7)]}.

Combine_Min method The *Combine_Min* method differs from *Combine_All* method in the fact than it only outputs high utility range Q-itemsets with minimal Q-intervals.

More precisely, the *Combine_Min* method follows the same traversing process of the *Combine_All* method. However, after generating a range HUQI, the *Combine_Min* method will immediately stop combining the current Q-itemset with the rest of Q-itemsets and it will directly pass to the next candidate Q-itemset in C.

Back to our example presented in Fig. 1, after generating [(A,2),(C,4,5)], there are two cases, if this Q-itemset is not a HUQI, the *Combine_Min* method continues with this Q-itemset by combining it with [(A,2),(C,6)] as the *Combine _All* method does. However, if [(A,2),(C,4,5)] is a HUQI, the *Combine_Min* method will move directly to combine the next Q-itemset [(A,2),(C,5)] with [(A,2),(C,6)] to generate [(A,2),(C,5,6)] without producing the remaining Q-itemsets presented in rectangles with rounded corners, i.e, Q-itemsets {[(A,2),(C,4,6)], [(A,2),(C,4,7)]}. Similarly, if [(A,2),(C,5,6)] is not a HUQI, the method combines it with [(A,2),(C,7)] and generates [(A,2),(C,5,7)]. If [(A,2),(C,5,6)] is a HUQI, the method moves directly to combine [(A,2),(C,6)] with [(A,2),(C,7)] without generating [(A,2),(C,5,6)].

After traversing all candidate Q-itemsets, the method performs the minimal Q-interval checking to keep only range Q-itemsets with minimal Q-intervals. Suppose that HUQIs resulting from the above process are: [(A,2),(C,4,6)], [(A,2), (C,5,6)] and [(A,2),(C,6,7)]. After performing the minimal Q-interval checking, the method will keep {[(A,2),(C,5,6)], [(A,2),(C,6,7)]} while Q-itemset [(A,2),(C,4,6)] will be discarded because the Q-interval of its last Q-item (C,4,6) is larger than the Q-interval of the last Q-item in [(A,2),(C,5,6)].

Combine_Max method The *Combine_Max* method does the contrary of the *Combine_Min* method. It outputs only high utility range Q-itemsets having maximal Q-intervals. More precisely, for each traversed candidate Q-itemset, the *Combine _Max* method keeps combining Q-itemsets as long as the conditions of the combining constraint presented in Definition 12 are verified. Following the same example depicted in Fig. 1, the method first combines [(A,2),(C,4)] with [(A,2),(C,5)] and Q-itemset [(A,2),(C,4,5)] is generated. Then, [(A,2),(C,4,5)] is combined with [(A,2),(C,6)] and [(A,2),(C,4,6)] is generated. Once again, [(A,2),(C,4,6)] is combined with [(A,2),(C,7)] to generate [(A,2),(C,4,7)]. The method does not combine [(A,2), (C,4,7)] with [(A,2),(C,4,9)] because the 3^{rd} condition of the combining constraint is not satisfied. After traversing the Q-itemset [(A,2),(C,4)], only one range Q-itemset is generated which is [(A,2),(C,4,7)].

At this point, the method moves to the next Q-itemset [(A,2),(C,5)] and repeats the above process. As a result, the Q-itemset [(A,2),(C,5,7)] is generated. The

method then moves to [(A,2),(C,6)] and so on until all Q-itemsets have been traversed. Similarly to $Combine\_Min$, $Combine\_Max$ checks the utility of the resulting Q-itemsets to keep only HUQIs. Moreover, after traversing all candidate Q-itemsets, the method performs the maximal Q-interval checking to keep only Q-itemsets with maximal Q-intervals.

Following our example, Q-itemsets generated by the traversing process using $Combine\_Max$ method are: [(A,2), (C,4,7)], [(A,2),(C,5,7)] and [(A,2), (C,6,7)]. By supposing that all generated Q-itemsets are HUQIs, the maximal Q-interval checking process will retain only Q-itemset [(A,2),(C,4,7)] which has the maximal Q-interval and other range Q-itemsets are eliminated.

In summary, the $Combine\_All$ method returns all possible patterns. The method $Combine\_Max$ allows identifying general patterns which can be seen as summarizing HUQIs generated by the $Combine\_All$ method. Whereas, $Combine\_Min$ provides small patterns which can be obtained by decomposing the patterns generated by the $Combine\_Max$ method.

3.3 Problem statement

Based on the previous definitions, the high utility quantitative itemset mining (HUQIM) problem is defined as follows:

Given a set of items $I = \{I_1, I_2, \ldots, l_N\}$ and a *quantitative transaction database* D that is composed of items from I, a *user-defined minimum utility threshold* θ and a *quantitative related coefficient* qrc, the problem of HUQIM is to find all exact Q-itemsets having a utility that is no less that θ as well as all range Q-itemsets that satisfy a user-selected combining constraint and have a utility no less than θ.

Formally, HUIM aims to find a set H that contains both high utility exact Q-itemsets, denoted as H_1, and high utility range Q-itemsets, denoted as H_2. The set H is defined by the following equation:

$$H = H_1 \cup H_2 \; such\; that \begin{cases} H_1 = \{X / X \cup I \; and \; u(X) \geq \theta\} \\ H_2 = \{Y / Y = Combine(CM, C, qrc) \; and \; u(Y) \geq \theta\} \end{cases} \quad (1)$$

where $Combine(CM, C, qrc)$ is a function that combines candidate Q-itemsets in C using one of the three combining methods ($Combine\_All$, $Combine\_Min$ or $Combine\_Max$) and qrc is a parameter for the combination constraint. The reader can refer to Section 3.2 to see how range q-itemsets are generated.

3.4 TWU pruning strategy for Q-itemsets

The TWU pruning strategy can be used to prune unpromising Q-itemsets during the mining process [18]. This strategy is based on the calculation of the transaction weighted utility (TWU) measure of Q-itemsets.

Definition 13 (TWU of a Q-itemset) The *transaction weighted utility (TWU) of a Q-itemset* X, denoted as $TWU(X)$, is the sum of utilities of transactions where X appears, that is $TWU(X) = \sum_{T_d \in OCC(X)} TU(T_d)$.

One important specificity of the TWU measure in HUQIM is that it is *anti-monotonic*. More precisely, given two Q-itemsets X and Y, if $X \subseteq Y$ then $TWU(X) \geq TWU(Y)$. Moreover, the TWU measure is an *upper bound on the utility* of Q-itemsets. In other words, $\forall (X \in D)$, $TWU(X) \geq u(X)$. Therefore, TWU can be used to prune unpromising Q-itemsets with their supersets.

Definition 14 (Promising Q-itemsets) Given a *user-defined minimum utility threshold* θ $(0 \leq \theta \leq \sigma)$, a *quantitative related coefficient* $(qrc > 0)$ and a Q-itemset X, X is a

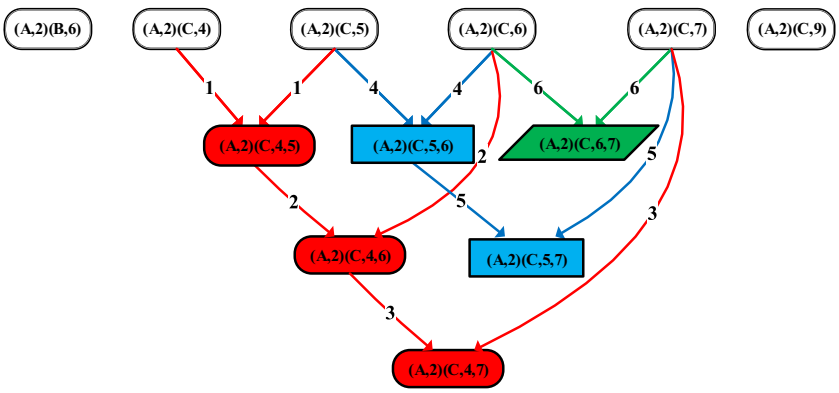

Fig. 1 Example of combining process

promising Q-itemset, if $TWU(X) \geq \frac{\theta}{qrc}$. Otherwise, X is an *unpromising Q-itemset*.

Property 1 (The TWU pruning strategy) TWU pruning strategy for Q-itemsets states that if a Q-itemset X is unpromising, then X is low utility Q-itemset as well as all its extensions.

4 Q-itemset utility-lists

This section introduces the utility-list structure, which is used by the proposed algorithm. A utility-list (UL) is a structure used to represent a pattern (Q-itemset in our case) and additional information about this pattern that is relevant for the pattern mining problem. A utility-list allows to quickly calculate the utility of its associated Q-itemset without the need to scan the database. Utility-lists (ULs) were initially used to mine high utility itemsets [23], and later have also been adopted for HUQIM [18, 19].

Utility-list uses an upper bound based on a function called remaining utility to prune the search space. Before calculating the remaining utility, Q-items in each transaction are first sorted according to the pre-defined total order relation $\prec$. More precisely, Q-items are first sorted according to their utilities in a descending order as suggested in [18, 19]. Then the remaining utility is calculated using the following definition.

Definition 15 (Remaining utility of a Q-itemset in transaction/ in the database) Let there be a set of distinct Q-items Q^* extracted from a database and sorted in each transaction according to a processing order $\prec$. The remaining utility of a Q-itemset X in a transaction $T_d \in OCC(X)$, denoted as $Rutil(X, T_d)$, is the sum of utilities of Q-items that come after all Q-items in X according to the $\prec$ order. Formally, $Rutil(X, T_d) = \sum_{x \in T_d/X} u(x, T_d)$

Here, T_d/X denotes the set of all Q-items that appear after all Q-items of X according to the $\prec$ order in T_d.

The remaining utility of a Q-itemset X in a database D is the sum of the remaining utilities of X in all transactions of its occurrence set. Formally, $Rutil(X) = \sum_{T_d \in OCC(X)} Rutil(X, T_d)$.

Considering the database D presented in Table 1, it is found that $Rutil(\{(A,2)\ (B,5)\},\ T_1) = u((C,2), T_1) + u((D,1), T_1) = 4 + 2 = 6$.

Definition 16 (Utility-list of a Q-itemset) The utility-list of a Q-itemset X, denoted as $UL(X)$, is composed of $|\ OCC(X)\ |$ tuples. Each tuple contains the utility information of X in one transaction in which X has appeared. Tuples have the form $\langle T_{id},\ Eutil,\ Rutil\rangle$, where T_{id} is the identifier of the transaction T_d, $Eutil(X, T_d)$ is the utility of X in T_d, i.e., $u(X, T_d)$, and $Rutil(X, T_d)$ is the remaining utility of X in T_d.

In addition to the list of tuples, the utility-list of a Q-itemset also stores the sum of all $Eutil$ values, denoted as $SumEutil$, which represents the exact utility of the Q-itemset and the sum of all $Rutil$ values, denoted as $SumRutil$, which will be used to prune low utility Q-itemsets.

Figure 2 shows the initial utility-lists for Q-items of the database presented in Table 1.

Using utility-lists allows to not only quickly calculate the utilities of Q-itemsets but also to prune more efficiently the search space using tighter pruning strategies compared with the TWU pruning strategy. More precisely, pruning the search space is performed by the following properties.

Property 2 ($SumEutil$ property) Given a Q-itemset X, $SumEutil$ in $UL(X)$ represents the exact utility of X, i.e, u(X). Therefore, if ($SumEutil < \theta$), then X is a low utility Q-itemset.

Fig. 2 Initial utility-lists for Q-items of database presented in Table 1

(A,2)	TWU: 41	
Tid	Eutil	Rutil
1	6	11
3	6	4
4	6	8
Sums	18	23

(B,4)	TWU: 10	
Tid	Eutil	Rutil
2	4	6
Sums	4	6

(B,5)	TWU: 17	
Tid	Eutil	Rutil
1	5	6
Sums	5	6

(B,6)	TWU: 14	
Tid	Eutil	Rutil
4	6	2
Sums	6	2

(C,2)	TWU: 27	
Tid	Eutil	Rutil
1	4	2
3	4	0
Sums	8	2

(C,3)	TWU: 10	
Tid	Eutil	Rutil
2	6	0
Sums	6	0

(D,1)	TWU :31	
Tid	Eutil	Rutil
1	2	0
4	2	0
Sums	4	0

Fig. 3 Constructing the utility-list of the 2-Q-itemset [(A,2),(D,1)]

(A,2)	TWU: 41	
Tid	Eutil	Rutil
1	6	11
3	6	4
4	6	8
Sums	18	23

(D,1)	TWU: 31	
Tid	Eutil	Rutil
1	2	0
4	2	0
Sums	4	0

⇨

((A,2),(D,1))	TWU: 31	
Tid	Eutil	Rutil
1	8	0
4	8	0
Sums	16	0

Property 3 (*SumRutil* pruning strategy) Given a Q-itemset X, if $(SumEutil + SumRutil < \theta)$, X and all its extensions are low utility Q-itemsets.

There are two fundamental operations that can be performed on utility-lists: The join operation and the merge operation. In the next subsections, these operations are presented.

4.1 Join utility-lists operation

The join operation is used to obtain utility-lists of larger Q-itemsets from smaller utility-lists. The initial case is to construct utility-lists of 2-Q-itemsets from those of initial Q-items (Definition 17). The general case is to construct utility-lists of (k+1)-Q-itemsets from utility-lists of k-Q-itemsets where $k > 2$ (Definition 18).

Definition 17 (Construction of the utility-lists of a 2-Q-itemset) Given two Q-items x and y where x is before y according to the $\prec$ order ($x \prec y$), the utility-list of Q-itemset $[xy]$ can be constructed from the intersection of tuples in $UL(x)$ with $UL(y)$. More precisely, common transactions are identified, i.e. common T_{id}, in $UL(x)$ and $UL(y)$. For each common transaction T_d, a new tuple E is added to $UL([xy])$ where T_{id} of E is the common T_{id}, $Eutil([xy], T_d) = Eutil(x, T_d) + Eutil(y, T_d)$ and $Rutil([xy], T_d) = Rutil(y, T_d)$.

Figure 3 shows an example of this operation where the utility-list of the Q-itemset [(A,2),(D,1)] is constructed from the utility-lists of Q-items (A,2) and (D,1).

Definition 18 (Construction of the utility-list of a (k+1)-Q-itemset) Let there be two k-Q-itemsets X, Y where $X = [(i_1, l_1, u_1), (i_2, l_2, u_2), \ldots, (i_{k-1}, l_{k-1}, u_{k-1}), (i_k, l_k, u_k)]$ and $Y = [(i_1, l_1, u_1), (i_2, l_2, u_2), \ldots, (i_{k-1}, l_{k-1}, u_{k-1}), (i_{k'}, l_{k'}, u_{k'})]$. If X and Y have the same prefix $P = [(i_1, l_1, u_1), (i_2, l_2, u_2), \ldots, (i_{k-1}, l_{k-1}, u_{k-1})]$ and the last Q-item $(i_k, l_k, u_k) \prec (i_{k'}, l_{k'}, u_{k'})$. The utility-list of $[XY] = [(i_1, l_1, u_1), (i_2, l_2, u_2), \ldots, (i_k, l_k, u_k), (i_{k'}, l_{k'}, u_{k'})]$ can be constructed from $UL(X)$, $UL(Y)$ and $UL(P)$. Besides, for each common transaction T_d in X and Y, a new tuple E is added into $UL([XY])$ where the T_{id} of E is the identifier T_{id} of transaction T_d, $util([XY], T_d) = Eutil(X, T_d) + Eutil(Y, T_d) - Eutil(P, T_d)$ and $Rutil([XY], T_d) = Rutil(Y, T_d)$.

Figure 4 shows an example where the utility-list of [(A,2),(C,2),(D,1)] is constructed from the utility-list of Q-itemset [(A,2),(C,2)], [(A,2),(D,1)] and that of the prefix [(A,2)].

4.2 Merge utility-lists operation

The merge operation is performed to obtain the utility-list of a range Q-itemset.

Definition 19 (Constructing utility-lists of range Q-itemsets) Let there be two k-Q-itemsets $X = [(i_1, l_1, u_1), (i_2, l_2, u_2), \ldots, (i_k, l_k, u_k)]$ and $Y = [(i'_1, l'_1, u'_1), (i'_2, l'_2, u'_2), \ldots, (i'_k, l'_k, u'_k)]$. If the following conditions hold:

(1) X and Y have the same prefix, i.e, $\forall j \in [1, k-1]$, $i_j = i'_j$, $l_j = l'_j$ and $u_j = u'_j$.

(2) For the last Q-item, $i_k = i'_k$ *and* $u_k = l'_k + 1$.

Then, X and Y can be merged to form a range Q-itemset $Z = [(i_1, l_1, u_1), (i_2, l_2, u_2), \ldots, (i_k, l_k, u'_k)]$. The utility-list $UL(Z)$ is obtained by merging tuples of $UL(X)$ and $UL(Y)$.

Figure 5 shows the $UL([B, 4, 5]]$ resulting from merging $UL([B, 4])$ with $UL([B, 5])$.

Fig. 4 Constructing the utility-list of the k-Q-itemset [(A,2),(C,2),(D,1)]

((A,2),(C,2))	TWU: 40	
Tid	Eutil	Rutil
1	10	2
3	10	0
Sums	20	2

((A,2),(D,1))	TWU: 31	
Tid	Eutil	Rutil
1	8	0
4	8	0
Sums	16	0

⇨

((A,2),(C,2),(D,1))	TWU: 17	
Tid	Eutil	Rutil
1	12	0
Sums	12	0

5 Proposed algorithm

This section describes the proposed FHUQI-Miner algorithm. But first, the section introduces a novel structure named TQCS (TWU of Q-items Co-occurrence based Structure) and the proposed search space pruning strategies that are based on TQCS.

5.1 The TQCS structure

The TQCS structure is composed of a set of tuples having the form (a, b, c) where a and b are two Q-items that have co-occurred in the database and $c \in \mathbb{R}$ is the TWU of the 2-Q-itemset obtained by their concatenation. i.e, $c = TWU([ab])$. TQCS is created by reading the database one time and similarly to the EUCS structure of FHM [10], TQCS is not represented as a matrix that contains the TWU of all possible 2-Q-itemsets. Alternatively, TQCS contains only TWU of all 2-Q-itemsets that really co-occur in the database. To this end, TQCS is composed of a set of tuples (a,b,c) such that $c \neq 0$. The reader can refer to the illustrative example of Section 5.4. Besides, Table 6 provides an example of the TQCS's construction for Q-items of the database presented in Table 3.

5.2 Proposed pruning strategies

The main advantage of the proposed FHUQI-Miner algorithm compared with the previous utility-list based algorithms [18, 19] is that FHUQI-Miner does not directly perform the join operation to form larger Q-itemsets. Alternatively, based on the TQCS, FHUQI-Miner adopts new pruning mechanisms that allow eliminating low-utility Q-itemsets with their extensions without the need to construct their utility-lists.

In contrast with the EUCS of FHM where there exists only one type of itemsets [10], the proposed TQCS deals with both exact and range Q-itemsets based on the following properties:

Property 4 (Exact Q-items Co-occurrence Pruning Strategy (EQCPS)) Given two Q-items x and y, EQCPS states that, if there is no tuple (a, b, c) such that $x = a$, $y = b$ and $c \geq \frac{\theta}{qrc}$, then the Q-itemset $[xy]$ is not a high utility Q-itemset and also all its extensions.

For example, we can see from the last row of Table 6 that $TWU([(C, 8), (A, 3)]) = 620$. If the minimum utility threshold θ is set to 1400 and $qrc = 2$, then Q-itemset [(C,8),(A,3)] is pruned with all its extensions because $TWU([(C, 8), (A, 3)]) < (\frac{1400}{2})$.

A limitation of this property is that, it is applicable only when Q-items x and y are both exact Q-items. However, in the case of range Q-items, Property 4 never returns any tuple because TQCS contains only tuples of exact Q-items. Therefore, another pruning strategy is designed to consider range Q-items.

If x is a range Q-item, we need to take into consideration all exact Q-items that are included in x (Definition 1). More precisely, the proposed strategy identifies TWU values of all Q-itemsets resulting from the combination of each exact Q-item included in x with Q-item y. TWU values can be easily found using the TQCS structure. According to the sum of TWU values, we can decide to consider or to prune the Q-itemset $[xy]$ using the following property.

Property 5 (Range Q-items Co-occurrence Pruning Strategy (RQCPS)) Given a range Q-item $x = (i, l, u)$ and an exact Q-item y, let $x_i = (i, q)$ be an exact Q-item extracted from x with quantity q where $l \leq q \leq u$ and let c_i be the TWU value of Q-itemset $[x_i y]$ taken from the TQCS structure. If $\sum_{i=l}^{u} c_i < \frac{\theta}{qrc}$, then Q-itemset $[xy]$ is a low utility Q-itemset as well as all its extensions.

For example, $TWU([(C, 7, 8), (H, 4)]) = TWU([(C, 7), (H, 4)]) + TWU([(C, 8), (H, 4)]) = 2840 + 0 = 2840$. If $\theta = 1400$ and $qrc = 2$, then Q-itemset [(C,7,8),(H,4)] is not pruned because $TWU([(C, 7, 8), (H, 4)]) \geq \frac{1400}{2}$.

5.3 FHUQI-Miner algorithm

This section presents the proposed FHUQI-Miner algorithm. The proposed algorithm follows the same procedure as algorithms proposed in [18, 19]. However, FHUQI-Miner adopts additional punning strategies to explore the search space more efficiently. FHUQI-Miner takes as input four parameters: (1) The transaction database D with quantities of items in different transactions (internal utilities) and profits of different items (external utilities), (2) The predefined minimum utility threshold (θ), (3) The Combining Method (CM) to be applied and (4) The quantitative related

(B,4)	TWU: 10	
Tid	Eutil	Rutil
2	4	6
Sums	4	6

(B,5)	TWU: 17	
Tid	Eutil	Rutil
1	5	6
Sums	5	6

⇨

(B,4,5)	TWU: 27	
Tid	Eutil	Rutil
1	5	6
2	4	6
Sums	9	12

Fig. 5 Constructing the utility-list of [(B,4,5)] by merging (B,4) and (B,5)

coefficient (qrc). The output is the set of all Q-itemsets having a high utility.

Algorithm 1 The FHUQI-Miner algorithm.

Input : D: The quantitative transaction database, θ: The user-defined minimum utility threshold, CM: The combining method ($Combine\_Min$, $Combine\_Max$ or $Combine\_All$), qrc: The quantitative related coefficient.
Output: The complete set of HUQIs.

1 First database scan to calculate the TWU of each Q-item;
2 Create initial set of promising Q-items P^* such that $\forall x \in P^* : TWU(x) \geq \frac{\theta}{qrc}$;
3 Second database scan to create utility-lists of promising Q-items $ULs(P^*)$ and build the TQCS structure;
4 **foreach** $x \in P^*$ **do**
5 **if** $UL(x).SumEutil \geq \theta$ **then**
6 $H = H \cup x$;
7 $Output\ x$;
8 **end**
9 **else**
10 **if** $UL(x).SumEutil + UL(x).SumRutil \geq \theta$ **then**
11 $E = E \cup x$;
12 **end**
13 **if** $\frac{\theta}{qrc} \leq UL(x).SumEutil \leq \theta$ **then**
14 $C = C \cup x$;
15 **end**
16 **end**
17 **end**
18 Discover High Utility range Q-itemsets (HR) using CM and C;
19 $QI_s \leftarrow sort(H \cup E \cup HR)$;
20 Recursive_Mining_Search($\emptyset$, QI_s, $ULs(QI_s)$, P^*, qrc, CM, θ);

FHUQI-Miner starts by scanning the database D for the first time to calculate the TWU values of initial Q-items using Definition 13 (line 1). Based on TWU values, the TWU pruning strategy (Property 1) is applied to prune unpromising Q-items early and to keep only promising Q-items whose extensions and combinations may be HUQIs (Definition 14). All promising Q-items are stored in the set P^* (line 2). After that, a total order relation $\prec$ on Q-items of P^* is established where $\prec$ is the descending order of Q-items utilities as suggested in [18, 19]. A second database scan is then performed. Besides, promising Q-items in each transaction are first reordered according to the $\prec$ order. Then, the utility-list of each promising Q-item is built. Moreover, the TQCS is built for Q-items of P^* (line 3).

After the construction of the TQCS, FHUQI-Miner first checks the utility of Q-items in P^* (lines 4 to 17). For each Q-item x in P^*, if $u(x) \geq \theta$, x is outputted as it is a HUQI and it is put in the set H which contains HUQIs (Definition 10). Otherwise, FHUQI-Miner performs two tests: (1) If $\frac{\theta}{qrc} \leq u(x) \leq \theta$, x is put in C set where C contains candidate Q-itemsets that can be combined together to form high utility range Q-itemsets (definition 11). (2) if $u(x) + UL(x).SumRutil \geq \theta$, x is put in the set E which contains all Q-itemsets that should be explored because one or more than one of their extensions may have high utilities (Property 3).

If the C set is not empty, the CM combining method is applied (line 18). The CM method tries to combine Q-itemsets of C to produce HR and $ULs(HR)$ where HR is the set of high utility range Q-itemsets that are generated by combining candidate Q-items of C (Definition 12) and $ULs(HR)$ are their corresponding utility-lists. Then, FHUQI-Miner creates the set QI_s which is composed of the union of Q-items in sets H, E and HR. Q-items in QI_s are reordered according to $\prec$ order (line 19).

At this point, FHUQI-Miner calls the $Recursive\_Mining\_Search$ procedure (line 20). The main steps of this procedure are illustrated in Algorithm 2.

The $Recursive\_Mining\_Search$ procedure is a recursive depth-first search algorithm that takes as input the following parameters: (1) The prefix Q-itemset P, (2) The set of Q-itemsets QI_s, (3) Utility-lists of Q-itemsets $UL_s(QI_s)$, (4) The list of promising Q-itemsets P^* (5) The quantitative related coefficient (qrc), (6) The combining method CM and (7) The minimum utility threshold (θ).

At the first call of the algorithm, the prefix Q-itemset P is $\emptyset$ and QI_s contains exact and range Q-items that are identified in Algorithm 1.

The $Recursive\_Mining\_Search$ algorithm operates as follows: For each extension $[Px]$ of P where $x \in QI_s$, the algorithm traverses all extensions $[Py]$ of P where $y \in P^*$ and $y \succ x$ to explore extensions of the form $[Pxy]$ (lines 1 to 3).

For each extension $[Pxy]$, the algorithm performs a pruning check based on TQCS structure to decide whether to consider Q-itemset $[Pxy]$ or to directly prune this Q-itemset without spending time on creating its utility-list (lines 4 to 15). There are two cases:

(1) Both Q-items x and y are exact Q-items (lines 4 to 9). In this case, property 4 is applied. More precisely, the algorithm searches the tuple (X, Y, c) in the TQCS structure. If $c = \emptyset$ or $c < \frac{\theta}{qrc}$, the algorithm will pass directly to the next extension Py without constructing $UL([Pxy])$ because the Q-itemset $[Pxy]$ with all its extensions are LUQIs (line 7).

(2) If x is a range Q-item (lines 10 to 15), then the algorithm applies property 5 to look for tuples in the TQCS structure that correspond to each exact Q-item included in x with Q-item y. If the sum of obtained TWU values, i.e, sum of extracted c values, of these tuples is less than $\frac{\theta}{qrc}$, the algorithm prunes the combination $[Pxy]$ without constructing its utility-list as in the first case (line 13).

For both cases, if $TWU([xy]) \geq \frac{\theta}{qrc}$, the algorithm performs the join process to build $UL([Pxy])$ from $UL(P)$, $UL([Px])$ and $UL([Py])$ (line 16).

Based on $UL([Pxy])$, if $[Pxy]$ is not promising, $[Pxy]$ is pruned and the algorithm will pass directly to another extension $[Py]$. Otherwise, the extension $[Pxy]$ is put in a new list of promising Q-itemsets P^* (line 18) and the algorithm performs similar tests as in Algorithm 1 to check if $[Pxy]$ is a HUQI (belongs to the H set), $[Pxy]$ is to be explored (belongs to the E set) or $[Pxy]$ is a candidate Q-itemset (belongs to the C set) (lines 18 to 28).

After traversing all extensions $[Py]$, the combination process is performed to extract high utility range Q-itemsets (HR) using the CM method (line 31). Then, the new set QI_s is formed from the union of H, E and HR (line 32).

At this point, the $Recursive\_Mining\_Search$ algorithm is recursively called with new prefix $[Px]$ and new QI_s and P^* with respect to prefix $[Px]$ (line 33). Since the algorithm starts from single Q-items and then recursively explores the search space by appending these single Q-items, the algorithm is able to discover the complete set of HUQIs.

5.4 An illustrative example

In this section, we give an example to illustrate how the designed FHUQI-Miner algorithm is applied. Consider the database presented in Table 3 with external utilities of different items shown in Table 4 and suppose that the $Combine\_All$ method is selected with $qrc = 5$ and θ=25%. Accordingly, the minimum utility threshold value is (TU x 0.25 = 5376 x 0.25 = 1344).

Table 3 An example of a transaction database

T_{id}	Transaction
T_1	(A,2) (C,7) (H,4) (I,9)
T_2	(A,3) (C,8)
T_3	(A,2) (B,1) (C,7) (G,7) (H,4)
T_4	(B,2) (C,9) (G,8) (H,5)
T_5	(A,2) (D,5) (E,1) (F,1)

Table 4 Profit table

Item	A	B	C	D	E	F	G	H	I
Profit	20	15	70	54	11	100	75	47	96

First, the database is scanned to calculate TWU values of the different Q-items. Based on TWU values, promising Q-items are identified. TWU values of all Q-items are presented in Table 5. We can see from Table 5 that, all Q-items are promising because their TWU values are greater than $\frac{\theta}{qrc}$ (Definition 14). Thus, all Q-items should be considered during the recursive mining process. The second database scan is then performed to construct utility-lists of all promising Q-items. Moreover, the algorithm also constructs the TQCS structure. The TQCS structure for this example is given in Table 6.

After that, promising Q-items are ordered based on their utilities in descending order. The ordered list of Q-items is P^* = {(I,9), (C,9), (G,8), (C,8), (G,7), (C,7), (D,5), (H,5), (H,4), (F,1), (A,3), (A,2), (B,2), (B,1), (E,1)}. At this point, the algorithm traverses Q-items of P^* to check the utility of each initial Q-item using its corresponding utility-list. Based on their utilities, Q-items are put in the set of HUQIs (H), the set of candidates Q-itemsets (C), or the set of Q-itemsets to be explored (E). After traversing all Q-items in P^*, E ={(I,9), (C,9), (C,7)}, C = {(I,9), (C,9), (G,8), (C,8), (G,7), (C,7), (D,5), (H,4)} while H set is empty because all Q-items in P^* are LUQIs.

FHQUI-Miner then tries to form high utility range Q-items with their utility-lists from the set C using the $Combine\_All$ method. From Q-items (C,7), (C,8) and (C,9), three range Q-items are generated which are (C,7,8), (C,8,9) and (C,7,9). Moreover, from Q-items (G,7) and (G,8), one range Q-item is generated which is (G,7,8). The utility of (C,7,8) (resp. (C,8,9), (C,7,9), (G,7,9)) is 1540 (resp. 1190, 2170, 1125). Thus, the algorithm outputs and keeps only high utility Q-items (C,7,8) and (C,7,9) while Q-items (G,7,8) and (C,8,9) are discarded. Accordingly, HR ={(C,7,8), (C,7,9)}.

Table 5 Promising items

Q-item	TWU	Q-item	TWU
(A,2)	3261	(B,1)	1258
(C,7)	2840	(G,7)	1258
(H,4)	2840	(A,3)	620
(I,9)	1582	(C,8)	620
(B,2)	1495	(D,5)	421
(C,9)	1495	(E,1)	421
(G,8)	1495	(F,1)	421
(H,5)	1495		

Algorithm 2 *Recursive_Mining_SearchAlgorithm.*

Input : P: The prefix Q-itemset, QI_s: The Q-itemsets list, $UL_s(QI_s)$: Utility lists of Q-itemsets, P^*: The list of promising Q-itemsets, qrc: The quantitative related coefficient, CM: The combining method, θ: The pre-defined minimum utility threshold

Output: The set of HUQIs with respect to prefix P.

foreach $[Px]$ *such that* $x \in QI_s$ **do**
  $QI_s \leftarrow \emptyset$; $P^* \leftarrow \emptyset$
  foreach $[Py]$ *such that* $y \in P^*$ *and* $y \succ x$ **do**
    if $[Px]$ *is an exact* $Q-itemset$ **then**
      $c \leftarrow TQCS(x, y)$;
      if $(c == Null\ Or\ c < \frac{\theta}{qrc})$ **then**
        Go to next $[Py]$;
      end
    end
    else
      $c \leftarrow \sum_{q=l}^{u} TQCS(x_i, y)$;
      if $c < \frac{\theta}{qrc}$ **then**
        Go to next $[Py]$;
      end
    end
    $Z \leftarrow [Pxy]$; $UL(Z) = Construct(x, y, P)$;
    if $UL(Z)! = Null\ and\ TWU(Z) >= \frac{\theta}{qrc}$ **then**
      $P^* = P^* \cup Z$; **if** $UL(Z).SumEutil \geq \theta$ **then**
        $H = H \cup Z$; *Output* Z;
      end
      else
        if $UL(Z).SumEutil + UL(Z).SumRutil \geq \theta$ **then**
          $E = E \cup Z$;
        end
        if $\frac{\theta}{qrc} \leq UL(Z).SumEutil \leq \theta$ **then**
          $C = C \cup Z$;
        end
      end
    end
  end
  Discover High Utility range Q-itemsets HR using CM and C;
  $QI_s \leftarrow (H \cup E \cup HR)$;
  Recursive_Mining_Search(Px, QI_s, $UL_s(QI_s)$, P^*, qrc, CM, θ);
end

Table 6 TQCS structure for Q-items of the database presented in Table 3

a	b	c	a	b	c
	(A,2)	2840		(E,1)	421
(C,7)	(B,1)	1258	(D,5)	(A,2)	421
	(H,4)	2840		(F,1)	421
(A,3)	ϕ	ϕ	(B,2)	ϕ	ϕ
(A,2)	(E,1)	421	(G,8)	(B,2)	1495
	(B,1)	1258		(H,5)	1495
	(A,2)	1582		(A,2)	1285
(I,9)	(C.7)	1582	(G,7)	(C,7)	1285
	(H,4)	1582		(B,1)	1285
(B,1)	ϕ	ϕ		(H,4)	1285
(H,5)	(B,2)	1495	(E,1)	ϕ	ϕ
(H,4)	(A,2)	2840		(G,8)	1495
	(B,1)	1258	(C,9)	(B,2)	1495
(F,1)	(E,1)	421		(B,5)	1495
	(A,2)	421	(C,8)	(A,3)	620

After performing the combination process, the algorithm forms the QI_s set from the union of H and E and HR. In this example, QI_s = {(I,9), (C,7,9), (C,9), (G,8), (C,7,8)}.

At this point, the *Recursive_Mining_Search* procedure is invoked to perform the depth-first search. The procedure recursively explores the search space starting from the first Q-item in QI_s until reaching the last Q-item. Besides, the join operation is used to form larger Q-itemsets. However, before joining Q-itemsets, the algorithm first applies the proposed EQCPS and RQCPS to avoid considering unpromising Q-itemsets.

Back to our example, the extension Px=(I,9) is processed with all extensions Py such that $y \in P^*$. (I,9) is first processed with extension Py =(C,9). From Table 6, we can see that there is no tuple $(a, b, c) \in TQCS$ such that a=(I,9), b=(C,9) and $c \geq (\frac{\theta}{qrc})$. Thus, Q-itemset [(I,9), (C,9)] is directly ignored and the algorithm will pass to the next Q-item $y \in P^*$ to test a new extension Py without constructing $UL([(I, 9), (C, 9)])$. Similarly, the algorithm passes directly extensions Py =(G,8), Py=(C,8) and Py=(G,7) until reaching Py =(C,7). From Table 6, we can see that $TWU([(I, 9), (C, 7)]) = 1582 \geq \frac{\theta}{qrc}$. Thus, Q-items (I,9) and (C,7) are joined to form [(I,9),(C,7)] and $UL([(I, 9), (C, 7)])$ is constructed. Once a new larger Q-itemset is constructed, the algorithm puts this Q-itemset in the P^* set and tests its utility to see if this Q-itemset is a $HUQI$ (belongs to H set), is to be explored (belongs to E set) or to be combined (belongs to C set). Since $u([(I, 9), (C, 7)]) = 1354$, The Q-itemset [(I,9), (C,7)] is outputted and it is put in the H set. The algorithm continues

traversing extensions Py. After traversing all extensions, H ={(C,7)}, C ={[(H,4),(A,2)]} and $E = \emptyset$.

The combination process is then performed on candidate Q-itemsets of C. Since the two Q-items in C cannot be combined, the combining process does not produce any range Q-itemset and $HR = \emptyset$. At this point, the new QI_s is created, $QI_s = \{H \cup E \cup HR\}$ = {(C,7) ∪∅ ∪ ∅} ={(C,7)}. Moreover, the new list of promising items is $P^* = \{$(C,7), (H,4), (A,2)$\}$.

At this point, the procedure is recursively called to explore larger Q-itemsets with new prefix P =(I,9), QI_s ={(C,7)}, P^* ={(C,7), (H,4), (A,2)}. The procedure continue in the same way until all HUQIs are found. The set of all HUQIs are presented in Table 7.

5.5 Complexity analysis

Lastly, this section discusses the complexity of the FHUQI-Miner algorithm. Let M and N be respectively the number of transactions and the number of distinct Q-items in the database. The algorithm starts by performing two database scans, the first scan is to calculate the TWU of Q-items. Besides, all transactions are scanned once where each transaction contains at most N Q-items. Thus, the first scan requires $O(MN)$ time.

The second database scan is performed to create initial utility-lists and the TQCS structure. To create initial-utility lists, a sort is performed on each transaction. This process has complexity $O(MNlog(MN))$. Moreover, to create the TQCS structure, it is necessary to check all co-occurrences in each transaction which requires $O(MN^2)$. Thus, the complexity of the two first database scans is $O(MN + MNlog(MN) + MN^2) = O(MN^2 + MNlog(MN))$.

The algorithm then checks the utility of each 1-Q-itemset which requires only $O(P^*)$. In the worst case, the complexity of this checking process is $O(N)$. After that, the algorithm performs the combining operation on candidate 1-Q-itemsets. The combining operation is performed on the set C of candidate Q-itemsets. Besides, Q-itemsets are first sorted based on their names. Then, all pairs of Q-itemsets in C are compared with each other. Thus, the complexity of the combining operation is $O(|C|^2 + |C|log(|C|)) = O(|C|^2)$. The complexity of the combining operation depends on the number of candidate Q-itemsets in C. The worst case corresponds to the situation where all 1-Q-itemsets are treated as candidates Q-itemsets. In that case, the complexity of the combining operation is $O(N^2)$. Hence, the total complexity before performing the recursive mining search is $O(MN^2 + MNlog(MN) + N^2) = O(MN^2 + MNlog(MN))$.

During the recursive search for patterns, FHUQI-Miner performs two main operations, the join operation and the combining operation.

The join operation is performed by recursively intersecting utility-lists of smaller Q-itemsets to get utility-lists of larger Q-itemsets. Given a prefix Q-itemset P and two extensions Px and Py such that $x \prec y$, the complexity of performing the join operation on Px and Py to obtain Pxy is $O((|Px| + |Py|)|P|)$ where $|P|$, $|Px|$ and $|Py|$ are the number of transactions in $UL(P)$, $UL(Px)$ and $UL(Py)$, respectively. In the worst case, the number of transactions in P, Px and Py is M. Accordingly, the complexity is $O(M^2)$.

The number of join operations to be performed depends on the pruning strategy that is utilized. Mathematically, the number of join operations is exactly the number of Q-itemsets that have not been pruned which equals the number of all possible Q-itemsets (2^N) minus the number of Q-itemset extensions that are pruned during the search for itemsets. Given that the number of Q-itemsets that are not pruned during the mining search is l_1, the overall complexity of join operations during the recursive search procedure is $O(l_1 M^2)$.

As mentioned above, the combination operation has a complexity $O(N^2)$ in the worst case. By assuming that the number of combining operations during the recursive mining search is l_2, the overall complexity of the combining process is $O(l_2 N^2)$.

Table 7 HUQIs for $\theta = 25\%$

Q-itemset	Utility
[(C,7,8)]	1540
[(C,7,9)]	2170
[(I,9),(C,7)]	1354
[(I,9),(C,7),(H,4)]	1542
[(I,9),(C,7),(A,2)]	1394
[(I,9),(C,7),(H,4),(A,2)]	1582
[(C,7,9),(H,4)]	1356
[(C,7,9),(A,2,3)]	1680
[(C,7,9),(G,8),(H,5)]	1465
[(C,7,9),(G,8),(H,5),(B,2)]	1495
[(C,7,9),(H,4),(A,2)]	1436
[(C,9),(G,8),(H,5)]	1465
[(C,9),(G,8),(H,5),(B,2)]	1495
[(C,7,8),(H,4)]	1356
[(C,7,8),(A,2,3)]	1680
[(C,7,8),(H,4),(A,2)]	1436
[(C,7),(H,4)]	1356
[(C,7),(H,4),(A,2)]	1436

In summary, the complexity of FHUQI-Miner is $O(MN^2 + MNlog(MN) + N^2 + l_1M^2 + l_2N^2) = (MN^2 + MNlog(MN) + l_1M^2 + l_2N^2)$.

It is worth noticing that pruning strategies that are integrated in the proposed algorithm are sufficiently effective to prune a large number of unpromising Q-itemsets when compared with HUQI-Miner. In fact, although HUQI-Miner and FHUQI-Miner theoretically have the same complexity, the number l_1 in our algorithm FHUQI-Miner is much smaller than l_1 in HUQI-Miner which makes FHUQI-Miner more efficient than the former algorithms as it will be demonstrated in the experimental evaluation of this paper.

6 Experimental results

This section describes experiments that were carried out to evaluate the performance of the proposed FHUQI-Miner algorithm. FHUQI-Miner is compared with the current state-of-art HUQIM algorithm, which is HUQI-Miner.

All experiments were performed on a PC equipped with an Intel(R) i7-8700 processor, 16 GB RAM and running the Windows7 operating system. The compared algorithms were implemented in Java by extending the SPMF Java open source data mining library [6].

Algorithms were tested using six different datasets, namely Foodmart, Retail, BMSWebView2, Pumsb, Mushroom and Connect. These datasets have various characteristics and they are commonly used to evaluate HUIM algorithms. All datasets were downloaded from the SPMF library [6]. However, except Foodmart, other datasets cannot be directly used for HUQIM since they provide utilities of items without giving their quantities. Thus, quantities and profits of different items were generated using the transaction database generator of SPMF [6]. Besides, profits of items were generated randomly between 1 and 10000 using a log-normal distribution while quantities were randomly generated from a pre-defined range for each dataset. The description of different parameters are presented in Table 8 while characteristics of each dataset based on these parameters are given in Table 9.

Table 8 Parameters description

	Parameters description
M	The number of transactions
N	The number of items
Q	Quantities range

Table 9 Characteristics of datasets

Dataset	M	N	Q	Type
Foodmart	4141	1559	1-10	Sparse
Retail	88162	16470	1-10	Sparse
BMSWebView2	77512	3340	1-10	Sparse
Mushroom	8416	128	1-10	Dense
Connect	67557	129	1-5	Dense
Pumsb	49046	2113	1-10	Dense

6.1 Execution time

In the first experiment, the execution time of FHUQI-Miner was compared with HUQI-Miner with different combining methods. We ran the two algorithms on each dataset with different values of θ. Results for both algorithms using $Combine\_All$, $Combine\_Min$ and $Combine\_Max$ are provided in Figs. 6, 7 and 8, respectively.

Results with *Combine_All* For the $Combine\_All$ method, it is observed in Fig. 6 that FHUQI-Miner clearly outperforms HUQI-Miner on four out of the six datasets used in these experiments. FHUQI-Miner is faster than HUQI-Miner for the Foodmart, Retail, BMSWebView2 and Pumsb datasets while the results are equivalent with a slight advantage for FHUQI-Miner for the Mushroom and Connect datasets.

For Retail, BMSWebView2, Foodmart and Pumsb, FHUQI-Miner is up to 22.18, 8.31, 4.79, 1.98 times faster than HUQI-Miner. On average, FHUQI-Miner is 12.21, 6.16, 3.39 and 1.58 faster than HUQI-Miner.

For the Mushroom and Connect datasets, the two algorithms have similar running time. However, FHUQI-Miner still outperforms HUQI-Miner in most of the cases. Globally, for the Mushroom and Connect datasets, FHUQI-Miner is on average 1.08 and 1.05 times faster than HUQI-Miner.

Results with *Combine_Min* It is observed in Fig. 7 that when the $Combine\_Min$ method is used, FHUQI-Miner is faster than HUQI-Miner on all datasets. More precisely, for Retail, BMSWebView2, Foodmart, Pumsb, Connect and Mushroom, FHUQI-Miner is up to 33.95, 14.72, 5.07, 2.05, 1.24 and 1.36 faster than HUQI-Miner. On average, FHUQI-Miner is 23.47, 12.53, 3.27, 1.83, 1.20 and 1.17 faster than HUQI-Miner.

Results with *Combine_Max* It is found in Fig. 8 that when utilizing $Combine\_Max$, FHUQI-Miner has better runtimes than HUQI-Miner on all datasets, and especially for Retail and BMSWebView2. In terms of results, FHUQI-Miner is up to 41.23, 18.77, 7.17 faster than HUQI-Miner for sparse datasets Retail, BMSWebView2 and Foodmart,

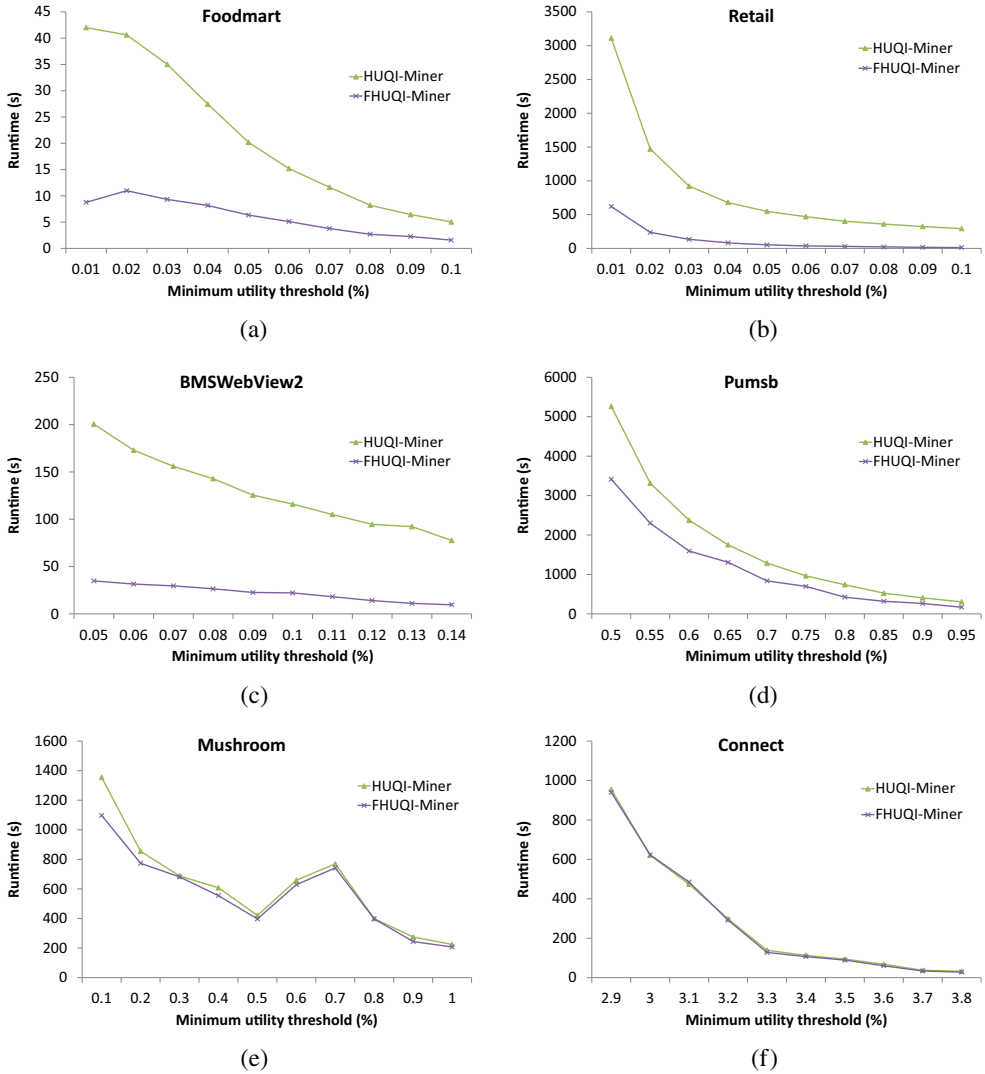

Fig. 6 Runtime with the *Combine_All* method

respectively. For dense datasets Pumsb, Mushroom and Connect, FHUQI-Miner is respectively up to 2,95, 1,58, 1,37 times faster than HUQI-Miner.

It is worth noticing that as θ is set to smaller values, more patterns may need to be evaluated by the algorithm. Thus, the search space becomes larger. In Figs. 6, 7 and 8, it can be observed that the difference between the two algorithms is bigger for small θ values. More precisely, the smaller the value of θ is, the faster FHUQI-Miner is compared to HUQI-Miner. Therefore, we can conclude that FHUQI-Miner is more efficient than HUQI-Miner when the search space becomes larger. This result will be further confirmed in next experiments.

6.2 Join operations

The search space of HUQIM can be represented as a graph in where a depth-first search algorithm os applied. Besides, the join operation which is performed to obtain larger Q-itemsets, can be viewed as visiting a new node in the search space. Accordingly, an efficient algorithm should perform as less join operation as possible to avoid visiting nodes that correspond to unpromising Q-itemsets.

In this experiment, the number of join operations for each dataset with different settings of θ was recorded.

The obtained results with *Combine_All*, *Combine_Min* and *Combine_Max* methods are respectively presented in Figs. 9, 10 and 11. Generally, it can be seen from these figures that for all combining methods, FHUQI-Miner performs much less join operations than HUQI-Miner for all datasets, especially sparse datasets.

For sparse datasets, the difference between the number of join operations of the two algorithms is huge especially with low values of θ. For example, on the Retail dataset with the *Combine_All* method and threshold $\theta = 0.01$, HUQI-Miner performs 1979170399 join operations while

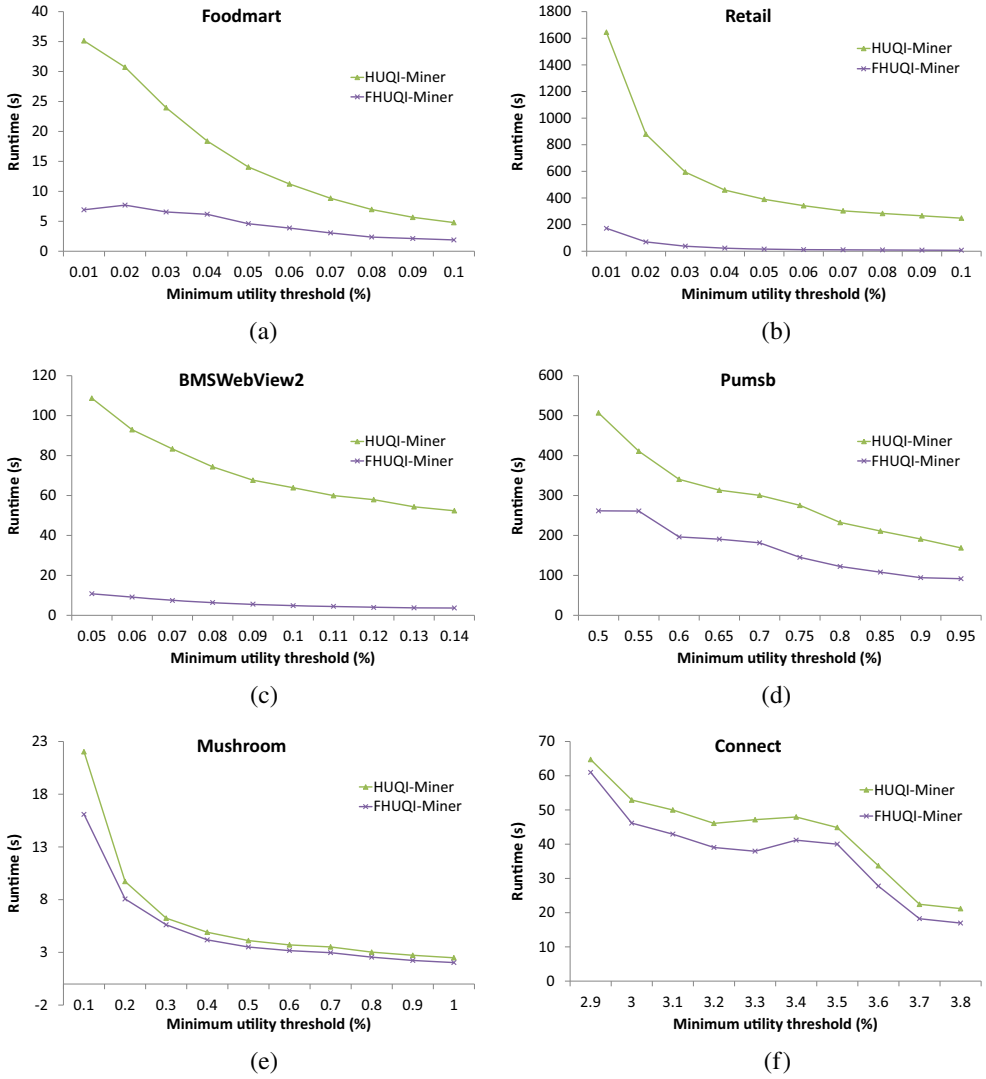

Fig. 7 Runtime with the *Combine_Min* method

FHUQI-Miner performs only 135424988 join operations. This means that HUQI-Miner processes 1843745411 unpromising Q-itemset which will be directly pruned by FHUQI-Miner without constructing their utility-lists. Similarly for Foodmart and a threshold 0.01, HUQI-Miner performs 60170299 join operations while FHUQI-Miner performs only 267979. The difference between the two algorithms is huge (59902320).

Similarly to when the *Combine_All* method is used, the number of join operations performed by FHUQI-Miner is much lower than HUQI-Miner when the *Combine_Min* and *Combine_Max* methods are applied. For instance, using the *Combine_Min* method, the difference between the number of join operations for BMSWebView2 with threshold $\theta = 0.05$ is 50412766 (Fig. 10-c). For the same dataset, when *Combine_Max* is applied, the difference is 78732 (Fig. 11-c).

For dense datasets, FHUQI-Miner also performs less join operations than HUQI-Miner. However, the gap between the two algorithms is not as large as for sparse datasets. Moreover, the performance of FHUQI-Miner on Pumsb is clearly better than its performance on the two other datasets, Mushroom and Connect. Starting with the *Combine_All* method, Pumsb and $\theta = 0.5$ (Fig. 9-d), HUQI-Miner performs 136344145 join operations while FHUQI-Miner only performs 93051302. Taking the same example, the difference of join operations for the *Combine_Min* and *Combine_Max* methods is respectively 6214571 and 3994845.

To clarify and summarize the obtained results, the average of differences between the two algorithms in terms of number of join operations for each dataset is calculated.

When adopting the *Combine_All* method, the difference between the two compared algorithms is 27068382.1

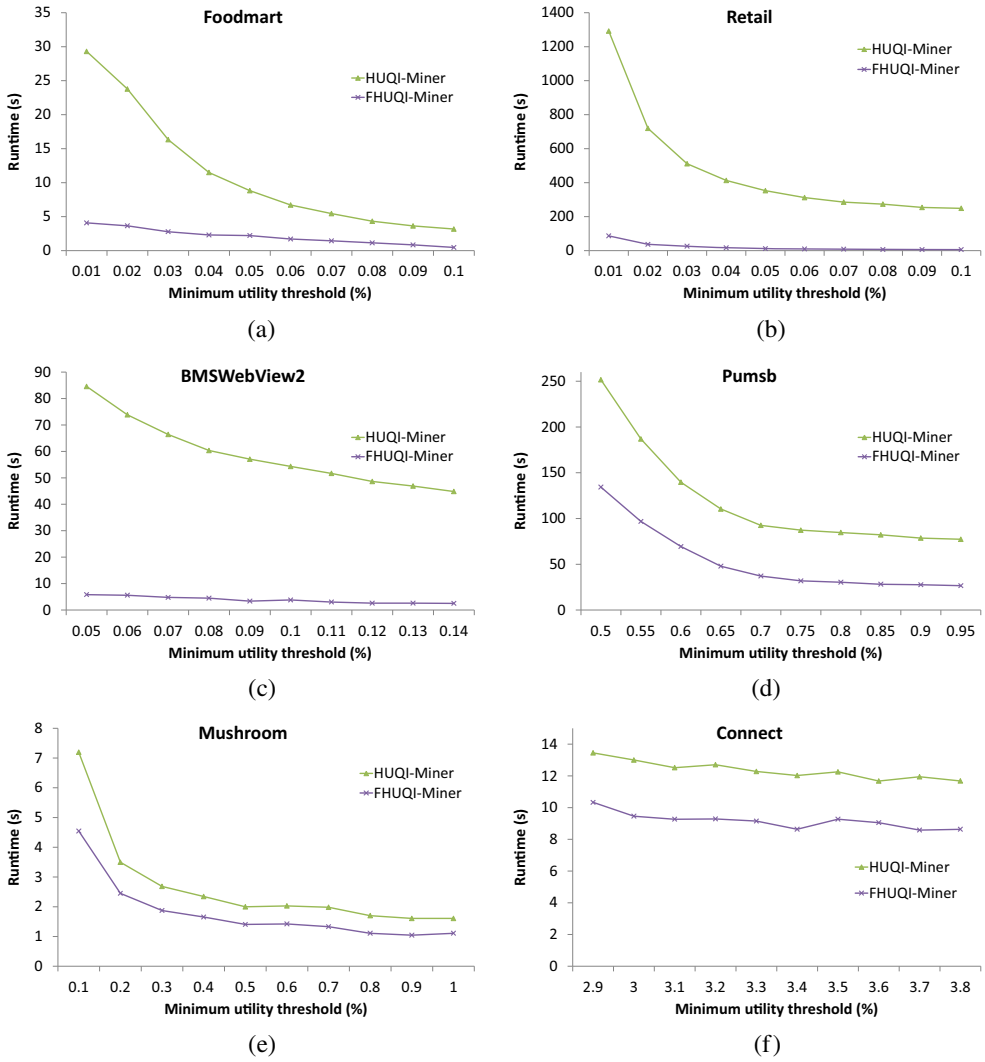

Fig. 8 Runtime with the *Combine_Max* method

(resp. 329932650.6, 42762584.2, 15639851.3, 21429504) for Foodmart (resp. Retail, BMSWebView2, Pumsb, Mushroom, Connect).

Using the *Combine_Min* method, the difference is 19633675,5 (resp. 197506293, 21884389,9, 3604549,5, 517312,1, 39730,5) for Foodmart (resp. Retail, BMSWebView2, Pumsb, Mushroom, Connect). Finally, with the *Combine_Max* method, the difference between the two compared algorithms is 15279436 (resp. 154308832,6, 15437347,8, 1574647,6, 332387,7, 17026,6) for Foodmart (resp. Retail, BMSWebView2, Pumsb, Mushroom, Connect).

6.3 Pruning strategies effectiveness

To demonstrate the effectiveness of the proposed pruning strategies, Tables 10 and 11 indicate the percentages of candidate Q-itemsets that FHUQI-Miner is able to directly prune that HUQI-Miner does not prune. These percentages are called pruning rates and they are calculated for different values of θ for both sparse and dense datasets.

For sparse datasets, it is clear from Table 10 that FHUQI-Miner is able to prune a large amount of unpromising candidate Q-itemsets when using all combining methods thanks to the proposed EQCPS and RQCPS pruning strategies, used during the search for patterns. More precisely, with the *Combine_All* method, FHUQI-Miner prunes up to 99.72%, 96.17% and 93.15% of unpromising Q-itemsets for the Foodmart, Retail and BMSWebView2 datasets, respectively. Using the *Combine_Min* method, FHUQI-Miner prunes up to 99.77%, 99.07% and 98.22% of candidate Q-itemsets for the Foodmart, Retail and BMSWebView2 datasets, respectively. Finally, with the *Combine_Max* method, FHUQI-Miner prunes

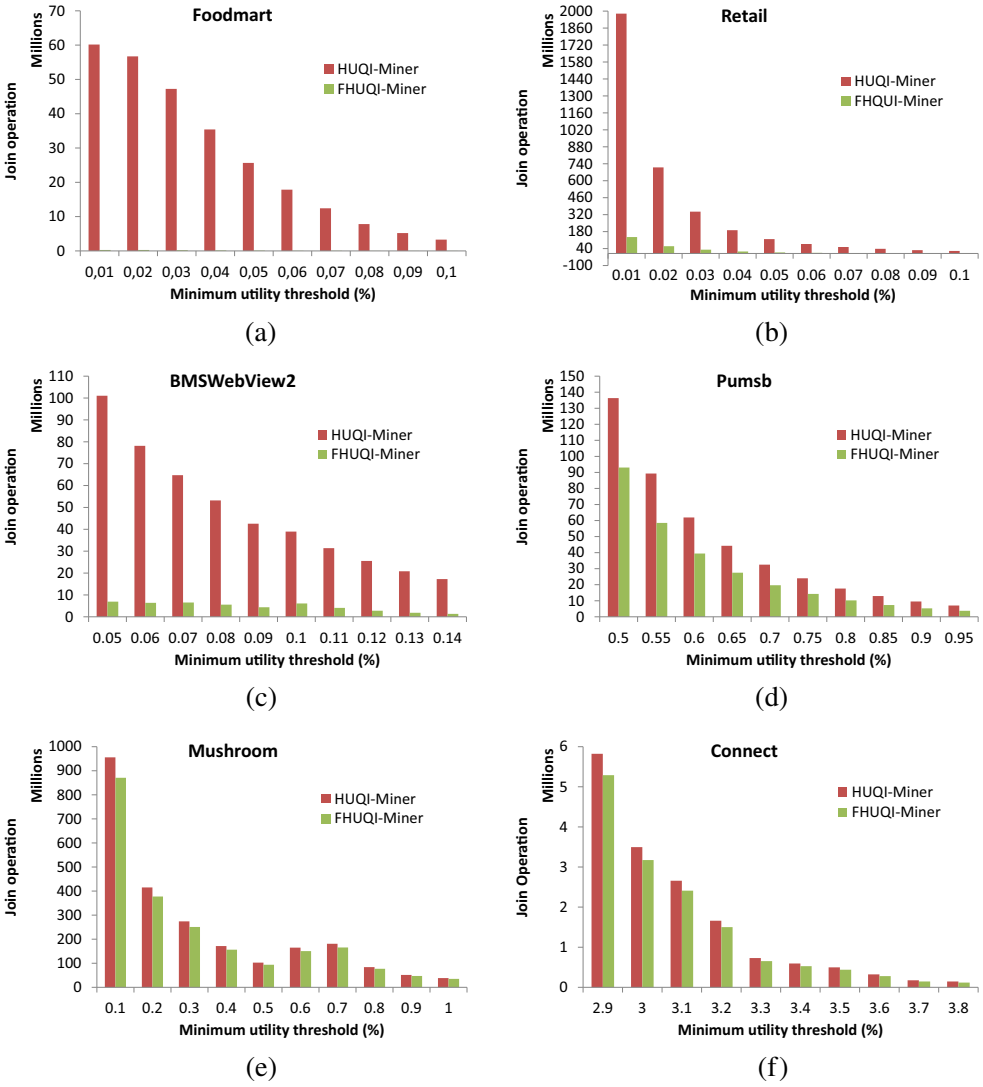

Fig. 9 Number of join operations with the *Combine_All* method

up to 99.82%, 99.43% and 98.88% of candidate Q-itemsets for the Foodmart, Retail and BMSWebView2 datasets.

By comparing the average pruning rates on different datasets with each combining method, it is found that, the largest average pruning rates are obtained on Foodmart, which is known as a very sparse dataset. On that dataset, 99.64%, 99.68%, and 99.69% of candidates are pruned when using the *Combine_All*, *Combine_ Min* and *Combine_Max* method, respectively. Retail is the second dataset in which pruning averages are large. Overall, the obtained results demonstrate the effectiveness of FHUQI-Miner on sparse datasets.

Pruning rates of FHUQI-Miner on dense datasets are given in Table 11. The highest pruning rates are obtained on Pumsb on which FHUQI-Miner can prune on average 40.43%, 39.23%, and 52.45% of candidate Q-itemsets with the *Combine_All*, *Combine_Min* and *Combine_Max* respectively. On the Connect dataset, FHUQI-Miner is able to prune 1,87%, 18,73% and 36,93% of candidates using *Combine_All*, *Combine_Min* and *Combine_Max* respectively. Finally, on Mushroom dataset, FHUQI-Miner can prune on average 8,64%, 28,74% and 48,13% with *Combine_All*, *Combine_ Min* and *Combine_Max*, respectively.

From the obtained results in terms of running time and pruning effectiveness, a natural question arises: Why the performance of FHUQI-Miner on sparse datasets is higher than its performance on dense datasets?

The major reason behind that is the characteristics of these datasets. Sparse datasets contain transactions that have few similarities. Accordingly, discovered HUQIs are not similar to each other. Moreover, the number of HUQIs is low and it is close to the number of distinct Q-

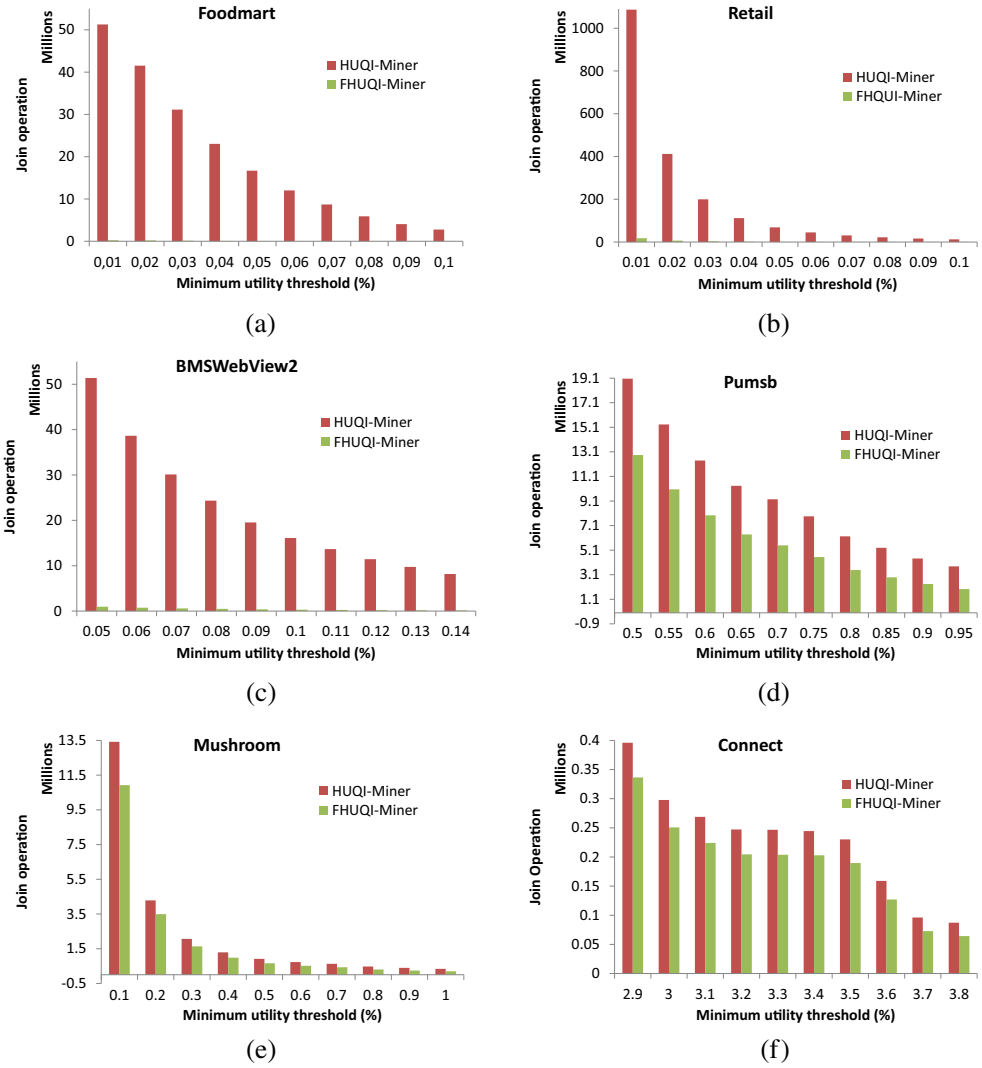

Fig. 10 Number of join operations with the *Combine_Min* method

items. For example, on Retail dataset and $\theta = 0.01$, the number of distinct Q-items is 101546 while the number of HUQIs is 89459. On BMSWebView2 with $\theta = 0.05$, the number of distinct Q-items is 101546 while the number of HUQIs is 89459. Therefore, a large part of Q-items combinations, i.e, Q-itemsets, are unpromising and they will be pruned during the recursive search for patterns (Algorithm 1) using the proposed EQCPS and RQCPS strategies. Consequently, the runtime is reduced, less join operations are performed, and more candidate itemsets are pruned.

On the other hand, transactions in dense datasets are similar and the number of HUQIs is very high comparing with the number of distinct Q-items. For instance, in Mushroom with threshold $\theta = 0.1$, the number of HUQIs is 8199402 while the number of distinct Q-items is only 1161. Due to this fact, many candidate Q-itemsets obtained by joining different Q-items during the recursive search for patterns are HUQIs. Accordingly, the frequency of successfully applying pruning strategies in dense datasets is lower than that for sparse datasets. As a result, the pruning rate of FHUQI-Miner on dense datasets is not as large as that for sparse datasets and the running time of the two algorithms is quite similar with an advantage for the proposed FHUQI-Miner algorithm.

It is worth noticing that, the high performance of FHUQI-Miner on sparse datasets is more advantageous since sparse datasets are more valuable and more common than dense datasets. In fact, most of real world problems such as customer behavior analysis have data sparsity. Therefore, obtaining better performance for mining patterns in sparse datasets is often viewed as more important than for dense datasets.

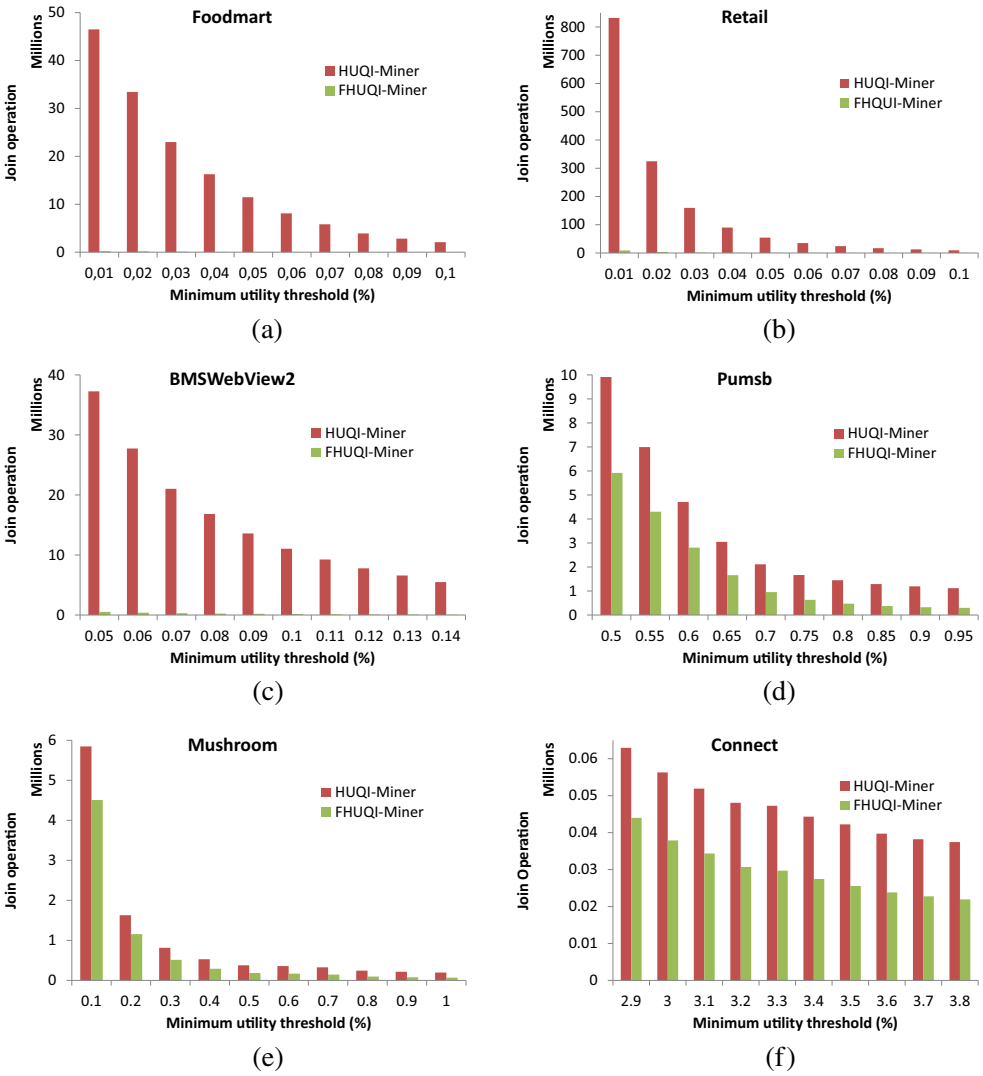

Fig. 11 Number of join operations with the *Combine_Max* method

Table 10 Percentage of candidate itemsets pruned by FHUQI-Miner for sparse datasets

Foodmart				Retail				BMSWebView2			
θ (%)	All (%)	Min (%)	Max (%)	θ (%)	All (%)	Min (%)	Max (%)	θ (%)	All (%)	Min (%)	Max (%)
0.01	99.55	99.54	99.52	0.01	93.15	98.35	98.88	0.05	93.15	98.11	98.59
0.02	99.50	99.52	99.48	0.02	91.58	98.30	98.83	0.06	91.84	98.05	98.55
0.03	99.55	99.58	99.54	0.03	90.92	98.43	98.89	0.07	89.88	97.99	98.53
0.04	99.61	99.64	99.63	0.04	91.49	98.59	98.99	0.08	89.52	97.97	98.55
0.05	99.67	99.71	99.71	0.05	92.94	98.76	99.11	0.09	89.72	97.99	98.57
0.06	99.70	99.75	99.76	0.06	93.27	98.86	99.18	0.1	84.33	98.01	98.61
0.07	99.70	99.76	99.80	0.07	94.36	98.98	99.29	0.11	87.00	98.04	98.67
0.08	99.71	99.76	99.81	0.08	94.86	99.03	99.34	0.12	89.09	98.12	98.74
0.09	99.72	99.77	99.82	0.09	95.82	99.05	99.40	0.13	91.05	98.15	98.80
0.1	99.72	99.76	99.82	0.1	96.17	99.07	99.43	0.14	92.18	98.22	98.88
Avg	99,64	99,68	99,69	Avg	93,46	98,74	99,13	Avg	89,78	98,07	98,65

Table 11 Percentage of candidate solutions pruned by FHUQI-Miner for dense datasets

Pumsb				Connect				Mushroom			
θ (%)	All (%)	Min (%)	Max (%)	θ (%)	All (%)	Min (%)	Max (%)	θ (%)	All (%)	Min (%)	Max (%)
0.5	31.75	32.60	40.30	2.9	9.14	15.03	30.10	0.1	8.89	18.61	22.90
0.55	34.47	18.91	8.63	3	9.18	15.80	32.70	0.2	9.08	18.50	28.96
0.6	36.31	23.23	7.88	3.1	9.36	16.63	33.82	0.3	8.46	20.81	36.93
0.65	37.92	38.27	45.56	3.2	9.55	17.30	36.12	0.4	8.86	24.11	44.78
0.7	39.44	40.57	54.74	3.3	10.59	17.37	37.14	0.5	8.73	27.65	51.12
0.75	40.53	42.18	62.11	3.4	11.15	17.01	38.08	0.6	8.68	29.85	52.71
0.8	41.89	44.08	67.42	3.5	11.85	17.62	39.46	0.7	8.43	31.43	55.55
0.85	43.18	45.40	70.83	3.6	13.29	20.06	40.01	0.8	8.35	35.67	60.14
0.9	44.51	47.04	72.71	3.7	16.30	24.30	40.44	0.9	8.38	39.00	63.15
0.95	46.52	48.92	73.35	3.8	18.28	26.22	41.42	1	8.55	41.73	65.05
Avg	40,43	39,23	52,45	Avg	11,87	18,73	36,93	Avg	8,64	28,74	48,13

7 Conclusion

In this paper, we have presented a new algorithm for high utility quantitative itemset mining named FHUQI-Miner (Faster High Utility Quantitative Itemset Mining). It relies on two new pruning strategies, named EQCPS (Exact Q-items Co-occurrence Pruning Strategy) and RQCPS (Range Q-items Co-occurrence Pruning Strategy). The proposed strategies can greatly reduce the number of join operations during the search for patterns which allows to improve the efficiency of HUQIM in terms of execution time. An extensive experimental study was performed on various datasets. Results have demonstrated the efficiency of the adopted pruning strategies especially for sparse datasets where the proposed algorithm is up to 41.23 times faster than HUQI-Miner. Moreover, the proposed algorithm can prune up to 99.82% of the search space.

There are many possibilities for extending this research in future work. First, we plan to propose new tighter upper bounds that can further improve the process of mining quantitative itemsets. Second, we plan to also integrate additional optimizations to speed up the processing of the existing combining methods which will make them faster. Third, another interesting orientation is to propose new combining methods that allow to discover more meaningful patterns in a database.

References

1. Ahmed CF, Tanbeer SK, Jeong BS (2010) A novel approach for mining high-utility sequential patterns in sequence databases. ETRI J 32(5):676–686
2. Aryabarzan N, Minaei-Bidgoli B (2018) Teshnehlab, M.: negfin: An efficient algorithm for fast mining frequent itemsets. Expert Syst Appl 105:129–143
3. Chan R, Yang Q, Shen YD (2003) Mining high utility itemsets. In: Proceedings of the 3rd IEEE international conference on data mining. IEEE computer society, pp 19–19
4. Dinh DT, Le B, Fournier-Viger P, Huynh VN (2018) An efficient algorithm for mining periodic high-utility sequential patterns. Appl Intell 48(12):4694–4714
5. Duong QH, Fournier-Viger P, Ramampiaro H, Nørvåg K., Dam TL (2018) Efficient high utility itemset mining using buffered utility-lists. Appl Intell 48(7):1859–1877
6. Fournier-Viger P, Gomariz A, Gueniche T, Soltani A, Wu CW, Tseng VS (2014) Spmf: a java open-source pattern mining library. J Mach Learn Res 15(1):3389–3393
7. Fournier-Viger P, Lin JCW, Duong QH, Dam TL (2016) Phm: mining periodic high-utility itemsets. In: Industrial conference on data mining. Springer, pp 64–79
8. Fournier-Viger P, Lin JCW, Truong-Chi T, Nkambou R (2019) A survey of high utility itemset mining. In: High-utility pattern mining. Springer, pp 1–45
9. Fournier-Viger P, Lin JCW, Vo B, Chi TT, Zhang J, Le HB (2017) A survey of itemset mining. Wiley Interdiscip Rev Data Min Knowl Discov 7(4):e1207
10. Fournier-Viger P, Wu CW, Zida S, Tseng VS (2014) Fhm: Faster high-utility itemset mining using estimated utility co-occurrence pruning. In: International symposium on methodologies for intelligent systems. Springer, pp 83–92
11. Fournier-Viger P, Yang P, Lin JCW, Yun U (2019) Hue-span: fast high utility episode mining. In: International conference on advanced data mining and applications. Springer, pp 169–184
12. Gan W, Lin JCW, Fournier-Viger P, Chao HC, Tseng VS, Yu PS (2018) A survey of utility-oriented pattern mining. arXiv:1805.10511
13. Gomariz A, Campos M, Marin R, Goethals B (2013) Clasp: An efficient algorithm for mining frequent closed sequences. In: Pacific-Asia conference on knowledge discovery and data mining. Springer, pp 50–61
14. Han J, Pei J, Kamber M (2011) Data mining: concepts and techniques. Elsevier
15. Han X, Liu X, Chen J, Lai G, Gao H, Li J (2019) Efficiently mining frequent itemsets on massive data. IEEE Access 7:31,409–31,421
16. Krishnamoorthy S (2017) Hminer: Efficiently mining high utility itemsets. Expert Syst Appl 90:168–183

17. Lee J, Yun U, Lee G, Yoon E (2018) Efficient incremental high utility pattern mining based on pre-large concept. Eng Appl Artif Intel 72:111–123
18. Li CH, Wu CW, Huang J, Tseng VS (2019) An efficient algorithm for mining high utility quantitative itemsets. In: 2019 international conference on data mining workshops (ICDMW). IEEE, pp 1005–1012
19. Li CH, Wu CW, Tseng VS (2014) Efficient vertical mining of high utility quantitative itemsets. In: 2014 IEEE international conference on granular computing (GrC). IEEE, pp 155–160
20. Lin CW, Hong TP, Lu WH (2011) An effective tree structure for mining high utility itemsets. Expert Syst Appl 38(6):7419–7424
21. Lin YF, Wu CW, Huang CF, Tseng VS (2015) Discovering utility-based episode rules in complex event sequences. Expert Syst Appl 42(12):5303–5314
22. Liu J, Wang K, Fung BC (2012) Direct discovery of high utility itemsets without candidate generation. In: 2012 IEEE 12th international conference on data mining. IEEE, pp 984–989
23. Liu M, Qu J (2012) Mining high utility itemsets without candidate generation. In: Proceedings of the 21st ACM international conference on information and knowledge management, pp 55–64
24. Liu Y, Liao WK, Choudhary A (2005) A two-phase algorithm for fast discovery of high utility itemsets. In: Pacific-Asia conference on knowledge discovery and data mining. Springer, pp 689–695
25. Luna JM, Fournier-Viger P, Ventura S (2019) Frequent itemset mining: A 25 years review. Wiley Interdiscip Rev Data Min Knowl Discov 9(6):e1329
26. Peng AY, Koh YS, Riddle P (2017) Mhuiminer: A fast high utility itemset mining algorithm for sparse datasets. In: Pacific-Asia conference on knowledge discovery and data mining. Springer, pp 196–207
27. Qu JF, Liu M, Fournier-Viger P (2019) Efficient algorithms for high utility itemset mining without candidate generation. In: High-utility pattern mining. Springer, pp 131–160
28. Truong-Chi T, Fournier-Viger P (2019) A survey of high utility sequential pattern mining. In: High-utility pattern mining. Springer, pp 97–129
29. Tseng VS, Wu CW, Shie BE, Yu PS (2010) Up-growth: an efficient algorithm for high utility itemset mining. In: Proceedings of the 16th ACM SIGKDD international conference on knowledge discovery and data mining, pp 253–262
30. Wang CM, Chen SH, Huang YF (2009) A fuzzy approach for mining high utility quantitative itemsets. In: 2009 IEEE International conference on fuzzy systems. IEEE, pp 1909–1913
31. Wang JZ, Huang JL, Chen YC (2016) On efficiently mining high utility sequential patterns. Knowl Inf Syst 49(2):597–627
32. Wu CW, Lin YF, Yu PS, Tseng VS (2013) Mining high utility episodes in complex event sequences. In: Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining, pp 536–544
33. Yen SJ, Lee YS (2007) Mining high utility quantitative association rules. In: International conference on data warehousing and knowledge discovery. Springer, pp 283–292
34. Yun U, Ryang H, Ryu KH (2014) High utility itemset mining with techniques for reducing overestimated utilities and pruning candidates. Expert Syst Appl 41(8):3861–3878
35. Zhang C, Han M, Sun R, Du S, Shen M (2020) A survey of key technologies for high utility patterns mining. IEEE Access 8:55,798–55,814
36. Zhang S, Wu X (2011) Fundamentals of association rules in data mining and knowledge discovery. Wiley Interdiscip Rev Data Min Knowl Discov 1(2):97–116
37. Zida S, Fournier-Viger P, Lin JCW, Wu CW, Tseng VS (2017) Efim: A fast and memory efficient algorithm for high-utility itemset mining. Knowl Inf Syst 51(2):595–625

Mourad Nouioua received his MSc degree of Computer science specialized in business intelligence from University of Bordj Bou Arreridj, Algeria in 2011 and 2013. He received the Doctor degree in Computer Science from Hunan University, China in 2019. Currently, he is a post-doctoral researcher at The Harbin Institute of Technology (Shenzhen). His research interests are: Data-Mining, Evolutionary computation and swarm intelligence.

Philippe Fournier-Viger is full professor at the Harbin Institute of Technology (Shenzhen). He has published more than 230 research papers in international conference proceedings and journals. He is also editor-in-chief of the Data Mining and Pattern Recognition journal, and director of the Center of Innovative Industrial Design. He is the founder of the popular SPMF open-source data mining library (http://www.philippe-fournier-viger.com/spmf/), which has been used in more than 600 research papers since 2010.

Cheng-Wei Wu received the PhD degree from the Department of Computer Science and Information Engineering, National Cheng Kung University, Taiwan, in 2015. Since 2015 to 2018, he worked as a postdoctoral researcher in the College of Computer Science, National Chiao Tung University, Taiwan. Currently, his is hired as an assistant professor in Department of Computer Science and Information Engineering, National Ilan University, Taiwan. He has published over 70 research papers and received more than 3,000 citations from Google Scholar Citations (http://goo.gl/8dokij). He is the author and the developer of the UP-Growth algorithm, which has been cited more than 850 citations. His current research interest includes data mining, utility mining, pattern discovery, big data analytics and artificial intelligence.

Jerry Chun-Wei Lin is currently working as an Associate Professor at Department of Computing, Mathematics and Physics, Western Norway University of Applied Sciences (HVL), Bergen, Norway. He has published more than 250 research papers in referred journals and international conferences. His research interests include data mining, soft computing, artificial intelligence, social computing, multimedia and image processing, and privacy-preserving and security technologies. He is also project leader of SPMF: An Open-Source Data Mining Library, and also serves as the Editor-in-Chief of the international journal of Data Science and Pattern Recognition.

Wensheng Gan received the B.S. degree in Computer Science from South China Normal University, Guangdong, China in 2013. He received the Ph.D. in Computer Science and Technology, Harbin Institute of Technology (Shenzhen), Guangdong, China in 2019. He was a joint Ph.D. student with the University of Illinois at Chicago, Chicago, IL, USA, from 2017 to 2019. He is currently an Association Professor with the College of Cyber Security, Jinan University, Guangzhou, China. His research interests include data mining, utility computing, and big data analytics. He has published more than 80 research papers in peer-reviewed journals (i.e., IEEE TKDE, ACM TKDD, ACM TOIT, ACM TDS, ACM TMIS, IEEE TCYB) and international conferences. He is an Associate Editor of Journal of Internet Technology.