Efficient mining of cross-level high-utility itemsets in taxonomy quantitative databases

N.T. Tung [a], Loan T.T. Nguyen [b,c,*], Trinh D.D. Nguyen [d], Philippe Fourier-Viger [e], Ngoc-Thanh Nguyen [f], Bay Vo [a]

[a] Faculty of Information Technology, HUTECH University, Ho Chi Minh City, Viet Nam
[b] School of Computer Science and Engineering, International University, Ho Chi Minh City, Viet Nam
[c] Vietnam National University, Ho Chi Minh City, Viet Nam
[d] Faculty of Information Technology, Industrial University of Ho Chi Minh City, Ho Chi Minh City, Viet Nam
[e] College of Computer Science and Software Engineering, Shenzhen University, Shenzhen, China
[f] Department of Applied Informatics, Wroclaw University of Science and Technology, Poland

ARTICLE INFO

Keywords:
Cross-level itemsets
High-utility itemsets
Taxonomy
Hierarchical database
Data mining

ABSTRACT

In contrast to frequent itemset mining (FIM) algorithms that focus on identifying itemsets with high occurrence frequency, high-utility itemset mining algorithms can reveal the most profitable sets of items in transaction databases. Several algorithms were proposed to perform the task efficiently. Nevertheless, most of them ignore the item categorizations. This useful information is provided in many real-world transaction databases. Previous works, such as CLH-Miner and ML-HUI Miner were proposed to solve this limitation to discover cross-level and multi-level HUIs. However, the CLH-Miner has a long runtime and high memory usage. To address these drawbacks, this study extends tight upper bounds to propose effective pruning strategies. A novel algorithm named FEACP (Fast and Efficient Algorithm for Cross-level high-utility Pattern mining) is introduced, which adopts the proposed strategies to efficiently identify cross-level HUIs in taxonomy-based databases. It can be seen from a thorough performance evaluation that FEACP can identify useful itemsets of different abstraction levels in transaction databases with high efficiency, that is up to 8 times faster than the state-of-the-art algorithm on the tested sparse databases and up to 177 times on the tested dense databases. FEACP reduces memory usage by up to half over the CLH-Miner algorithm.

1. Introduction

Data mining plays an important role in the field of knowledge discovery. By scanning through a huge amount of data, data mining algorithms can reveal useful pieces of information to users. The discovered knowledge, called itemsets, can help understand the data and support decision-making for many real-world applications [1]. Different types of itemsets can be found in various types of data. One of the most popular tasks of itemset discovery is frequent itemset mining (FIM, also known as frequent pattern mining) [2]. Algorithms for that task find all sets of items that have an occurrence frequency

* Corresponding author.
E-mail addresses: nt.tung@hutech.edu.vn (N.T. Tung), nttloan@hcmiu.edu.vn (L.T.T. Nguyen), 20126291.trinh@student.iuh.edu.vn (T.D.D. Nguyen), philfv@hit.edu.cn (P. Fourier-Viger), ngoc-thanh.nguyen@pwr.edu.pl (N.-T. Nguyen), vd.bay@hutech.edu.vn (B. Vo).

(*support*) that is no less than a user-specified minimum support threshold in a transaction database. Apriori is a well-known algorithm that was developed for this task [3]. Although it applies search space pruning properties to accelerate the itemset mining process, Apriori performs poorly in many scenarios as it scans the database multiple times and can join numerous itemsets to obtain larger itemsets. Several algorithms were subsequently designed to more efficiently discover frequent itemsets, such as FP-Growth [4], Eclat [5], and LCM [6], which can achieve a considerable performance improvement. After mining frequent itemsets, these latter methods can be used by a process called association rule mining (ARM) to derive association rules indicating strong associations between items.

Although FIM and ARM have been applied in many domains, they find itemsets mostly based on their occurrence frequencies, and ignore other factors that can help evaluate an itemset's importance such as its weight and profit. Based on this observation, the problem of FIM was generalized as high-utility itemset mining (HUIM) [7,8]. In HUIM, each item contained in a transaction is associated with two positive numeric values: an internal utility (or purchase quantity) and an external utility (or unit profit). HUIM's main goal is to identify the sets of items that have a high utility (yield the most profit) in a quantitative transaction database.

As with FIM, HUIM utilizes a user-specified threshold to filter uninteresting itemsets. It is named the minimum utility threshold and represents the least utility that an itemset must have to be considered a high utility itemset (HUI). Although HUIM is a generalization of FIM, HUIM is more challenging than FIM since the utility function used to evaluate the importance of an itemset does not satisfy the downward closure property (also called the anti-monotonicity property) [9]. This means that an itemset may have a utility that is smaller, equal, or greater than that of its subsets. Because efficient FIM algorithms rely on the downward closure property to reduce the search space of itemsets, they cannot be directly applied for HUIM. Hence, several dedicated HUIM algorithms were designed, such as Two-Phase [9], IHUP [10], UP-Growth [11], HUI-Miner [12], d^2HUP [13,14], FHM [15], EFIM [16], HMiner [17] and iMEFIM [18]. Although most of those algorithms are efficient, they ignore the fact that items in transaction databases are often organized into categories and subcategories of a *taxonomy*. For instance, Fig. 1 shows a taxonomy of items sold in a computer store. The items *Printer* and *Scanner* are categorized as *Peripherals*, while *Laptop* and *Desktop* are generalized as *Computers*. Moreover, *Peripherals* and *Computers* are subcategories (specializations) of the more general *Hardware* category. A taxonomy can play an important role in mining itemsets in a database, as it allows grouping items semantically in *generalized itemsets*, that is itemsets containing generalized items (categories). In Fig. 1, the generalized items are *Hardware*, *Computers* and *Peripherals*. Since traditional HUIM algorithms ignore taxonomy information, they can only discover itemsets containing items appearing at the lowest level of the taxonomy tree of a transaction database (*Printer, Scanner, Laptop* and *Desktop*). Thus, items such as *Peripherals*, *Computers* and *Hardware* cannot be found in the output of these algorithms, even if they are HUIs.

Based on FIM and ARM, several algorithms have been designed that take taxonomy information into account and have a wide range of applications [19–26]. Since the support (occurrence frequency) of itemsets is anti-monotonic, the proposed approaches can efficiently find generalized itemsets. However, integrating taxonomy information in HUIM is much more challenging, as the utility function does not respect the downward-closure property, especially regarding mining HUIs at different levels of abstraction. Recently, Cagliero et al. presented an algorithm named ML-HUI-Miner [27] to extract a novel itemset type called Generalized HUIs (GHUIs) in transaction databases, that is itemsets that yield a high profit and can be from different abstraction levels. In 2020, Fournier-Viger et al. defined a more general problem of mining cross-level HUIs [28] and an algorithm named CLH-Miner [28]. CLH-Miner addresses some limitations of previous studies by proposing new upper bounds on the utility and effective pruning strategies while allowing an itemset to mix items from different abstraction levels. This allows finding interesting itemsets that ML-HUI Miner cannot discover. However, CLH-Miner has a

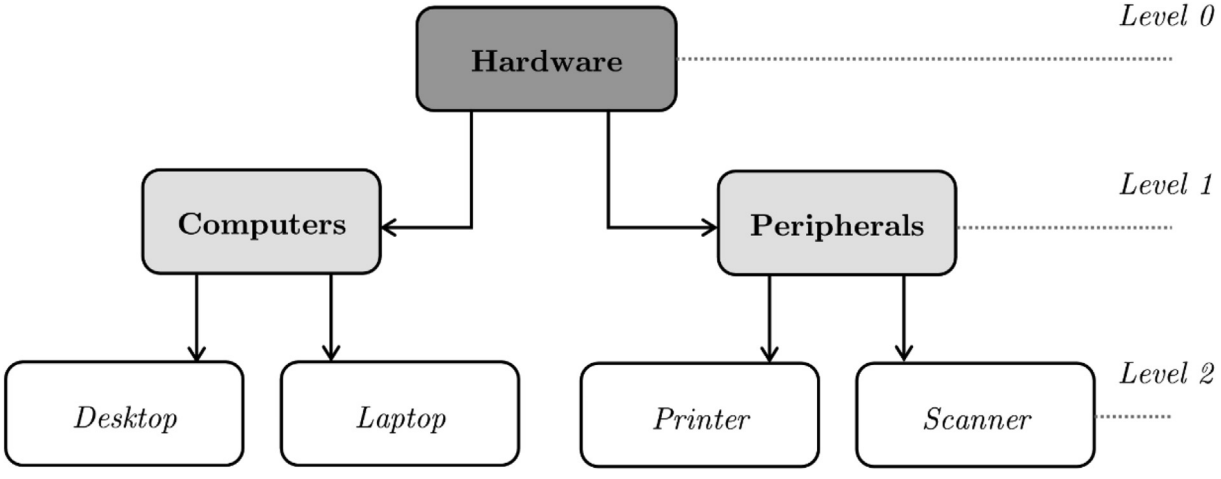

Fig. 1. A taxonomy tree of hardware devices.

considerably longer execution time and consumes much more memory than ML-HUI Miner and HUI-Miner. To tackle these challenges, this article presents a novel algorithm, named FEACP (Fast and Efficient Algorithm for Cross-level high-utility Pattern mining), which adopts effective pruning strategies to mine cross-level HUIs.

This study is motivated by two observations about CLH-Miner. The first one is that CLH-Miner uses two upper bounds (the remaining-utility and GTWU) to determine if an itemset should be extended, and thus reduce the search space. Although these two upper bounds were used in several studies, a major drawback is that they are not tight enough. Hence, many unpromising itemsets may not be eliminated from the search space. The second observation is that a modified version of the original utility-list structure is used by CLH-Miner, namely the tax-utility-list. This structure has a high computational cost for maintaining and extending itemsets.

The main contributions of this paper can be summarized as follows:

- Based on the concept of item taxonomy and item generalization, novel pruning strategies are presented, which rely on a series of upper bounds and properties to discard unpromising itemsets [13,14,16]. They are also mathematically proved to safely reduce the search space, which can considerably reduce the runtime and memory consumption.
- A method is proposed to compute the utility of generalized items at the same time as specialized items. It also helps to accurately re-evaluate the upper bounds to prune cross-level candidates.
- An efficient algorithm is introduced, named FEACP, which relies on all the proposed techniques to mine cross-level HUIs in taxonomy quantitative databases.

The remaining content of this paper is organized as follows. Section 2 reviews the literature about HUIM. Section 3 presents the preliminaries and defines the problem of mining cross-level HUIs. Section 4 describes the proposed candidate pruning strategies and the FEACP algorithm for mining cross-level HUIs. Section 5 then reports the results from an extensive experimental evaluation. Finally, a conclusion is drawn and plans for future work are discussed in the last section.

2. Related work

Since the proposal of FIM in transaction databases, several algorithms have been presented to efficiently solve the problem [1,2,4], and these exploit the downward closure property of the support measure to reduce the search space. However, FIM has the major drawback in that it ignores the importance as well as the purchase quantities of items in transactions. Hence, FIM can find many itemsets that frequently appear in transactions but yield a low profit, while ignoring many infrequent itemsets that yield a tremendous profit.

This major drawback of FIM has been addressed by generalizing FIM to the more general problem of HUIM [7]. It consists of finding itemsets with a high importance in terms of criteria such as weight or profit in transaction databases. As noted above, HUIM is far more challenging than FIM because the utility measure does not satisfy the downward closure property. Thus, a proper framework must be designed to effectively reduce the search space for mining HUIs. Several algorithms have been introduced to efficiently solve the HUIM problem [29–31]. The first correct and complete HUIM algorithm was Two-Phase [9], which introduced an upper bound on the utility, named Transaction Weighted Utility (*TWU*). It makes it possible to safely ignore unpromising itemsets to accelerate the mining process [9]. However, Two-Phase can still have a long runtime because it performs multiple database scans and can generate unpromising candidate itemsets. To further improve the efficiency of HUIM, several algorithms extended the pattern growth model proposed by Han et al. [4], such as IHUP [10] and UP-Growth [11]. Although they avoid the problem of generating unpromising candidates, these algorithms rely mainly on the *TWU* measure for search space reduction. A problem with this measure is that it is considered a loose upper bound on the utility, and thus often cannot reduce large parts of the search space.

In order to address this problem, several more effective pruning strategies and upper bounds were presented. In 2012, the HUI-Miner algorithm introduced the concepts of the remaining utility upper bounds and utility-list structure [12]. At about the same time, Liu et al. introduced the d^{2}HUP [13] algorithm, which relies on the concept of database projection and upper bounds such as the remaining utility and local utility and their pruning strategies. But in that paper, these upper bounds were defined using different terminologies. These upper bounds and pruning strategies were later discussed in more detail in a subsequent paper [14]. Without the need for a candidates generation phase, HUI-Miner and d^{2}HUP are the first two single-phase HUI mining algorithms [12]. Extending HUI-Miner [12], Fournier-Viger et al. designed the FHM algorithm [15] and proposed an effective pruning strategy named EUCP. Krishnamoorthy presented the HUP-Miner algorithm [32], which extends HUI-Miner [12] with two new and efficient pruning techniques. The EFIM algorithm [16] presented two new utility-based upper bounds, namely local and sub-tree utility. These upper bounds, in a way, are equivalent to the *uBitem* bound and *uBfpe* bounds in [13,14], respectively. With regard to the upper bounds of EFIM and those in previous works, the local utility upper bound is generally tighter than the TWU, but equivalent to the TWU for any itemset containing a single item [16]. The local utility bound is equivalent to the *uBitem* upper bound of d^{2}HUP [13,14], and the sub-tree utility upper bound is equivalent to the remaining utility upper bound and the *uBfpe* bound, which were proposed in HUI-Miner [12] and d^{2}HUP [13,14]. But in practice, how the upper bounds are applied by these algorithms in the process of high utility itemset mining is not the same. For instance, it was shown that the sub-tree utility of EFIM is applied earlier than the remaining utility of HUI-Miner, and can thus prune more itemsets. The reader can refer to [16] for a discussion of the upper bounds of EFIM.

Moreover, EFIM introduced two novel and effective techniques to reduce the cost of database scans and improve the effectiveness of HUIM, namely high-utility database projection (HDP) and high-utility transaction merging (HTM). It was found that EFIM consumes less memory and has almost linear complexity with the number of transactions. Nguyen et al. presented an algorithm, called iMEFIM [18], which extends EFIM to further reduce the cost of database scans. The authors also proposed a new utility framework to properly handle transaction databases containing items with dynamic profit values [33].

Recently, the HUIM problem has been extended in various ways to consider additional constraints. Fournier-Viger et al. introduced the ideas of local HUIs and peak HUIs to find interesting itemsets in non-predefined time intervals, and two algorithms named LHUI-Miner and PHUI-Miner to find these itemsets [34]. Vo et al. introduced an algorithm called CoHUI-Miner [35], which considers the relationship between items in high-utility itemsets to mine correlated HUIs (CoHUIs) in one phase. An algorithm called LTHUI-Miner was designed to identify HUIs that are trending during a period of time [36]. It can determine the time intervals where itemsets have an increasing or decreasing utility to discover insightful HUIs. Sahoo et al. extended the HUIM problem to discover high-utility association rules (HAR) [37] by proposing the HGB-HAR algorithm. Then, Mai et al. [38] developed the LARM algorithm to mine HARs more efficiently using a high-utility itemset lattice. After this, Mai et al. presented the LNR-HAR algorithm to mine non-redundant HARs based on the high-utility itemset lattice [39]. To remove the need for specifying or adjusting the minimum utility threshold, the problem of top-k high-utility itemset mining was studied and efficient algorithms were designed, such as TKU and TKO [40], TKUL-Miner [41], TONUP [42], TKEH [43] and THUI [44]. These approaches only require setting a small number of desired itemsets as k or n instead of a minimum utility threshold. Although HUIM is a very active research area, none of the above algorithms consider that items are organized in a taxonomy. However, taxonomy information is available in many practical real-life applications of itemset mining [23,26,45–48] and can provide valuable insights.

The concepts of item categorization, generalization, and taxonomy were introduced in the early days of research on FIM and ARM by Skirant and Agrawal [25]. They designed Cumulate, the first algorithm to mine association rules in a database with a taxonomy. Cumulate can find rules containing items from different abstraction levels in a transaction database. Later, Hipp et al. created a depth-first-search (DFS) algorithm named Prutax [49] to efficiently discover cross-level frequent itemsets by using a vertical database representation and two pruning strategies based on the taxonomy to eliminate infrequent candidate itemsets. Sriphaew and Theeramunkong proposed an algorithm named SET [50], which performs a set-enumeration based search and applies various constraints on the generalized itemsets to speed up the discovery process. Vo and Le developed a novel algorithm named MMS_GIT-tree [20], which requires only one database scan to mine generalized association rules. The authors presented an improved version of the IT-tree structure [51], named GIT-tree, to store hierarchical databases. The algorithm also put a structure called tidsets into use for faster support calculation of the generalized itemsets. Wu et al. designed the GMAR and GMFI algorithms to mine generalized association rules [21]. While GMFI generates generalized ARs by finding all frequent itemsets, GMAR directly constructs generalized ARs. Both methods adopt the FCET (frequent closed enumeration table) structure. Based on the FP-tree structure proposed by Han et al. [4], Pramudiono et al. proposed the FP-tax algorithm [52] to mine multi-level association rules by performing a bidirectional traversal of the constructed tree. Both traversal methods were shown to outperform Cumulate. Baralis et al. proposed a framework, namely CoGAR [24], which employs multiple taxonomies to mine generalized association rules using constraints, such as schema and opportunistic confidence constraints. Han et al. extended the multilevel itemset mining problem by combining it with multiple minimum support thresholds [19], such that a threshold can be assigned to each abstraction level of the taxonomy. The authors introduced a top-down breadth-first search (BFS) based approach to recursively analyze the transaction database, starting from the most abstract itemsets. Lui et al. presented a method to mine multi-level itemsets using multiple minimum support thresholds [53]. The proposed algorithm considers taxonomy distance and scans the original transactions $k+1$ times, where k is the length of the largest itemset. For multilevel/cross-level HUI mining, there are to the best of our knowledge only two algorithms that take an item taxonomy into account. The first one is ML-HUI Miner [27], which introduced the concept of generalized high-utility itemsets using a taxonomy. ML-HUI Miner is based on the FHM algorithm as it prunes candidates using the EUCS structure [27]. However, the relationships between different abstraction levels are not considered to further reduce the search space. The second algorithm, which was proposed recently by Fournier-Viger et al., is named CLH-Miner [28]. The authors defined a more general problem of mining cross-level HUIs to address some issues of ML-HUI Miner. Recently, a variation of CLH-Miner was also presented for mining the top-k cross-level HUIs, named TKC [54]. Although CLH-Miner extends FHM and introduces a novel GWU (generalized-weighted utilization) upper bound, it still suffers from long runtime and high memory usage.

To address these limitations of recent algorithms, this paper presents a novel algorithm for cross-level HUIM. But before presenting the algorithm, the next section introduces the preliminaries and the formal problem definition.

3. Preliminaries

The type of databases considered in this paper are quantitative transaction databases with a taxonomy. Though transaction databases are used in retail stores, their structure is quite general and can be used to represent data in many other applications. The concepts of transaction databases and taxonomy are formally defined as follows.

Definition 1. ***(Transaction database)*** *[28].* Let there be a set of items $\mathscr{I} = \{i_1, i_2, ..., i_m\}$. A quantitative transaction database is a multiset of transactions $\{T_1, T_2, ..., T_n\}$, denoted as $\mathscr{D}$. Each transaction T_j in $\mathscr{D}$ is a set of items (*i.e.*, $T_j \subseteq \mathscr{I}$) and has a unique transaction identifier (TID) j. Every item $v \in \mathscr{I}$ is associated with a positive integer indicating its relative importance (e.g., unit profit), called its external utility, and denoted as $p(v)$. For each item $v \in T_j$, a positive integer $q(v, T_j)$ is defined called the internal utility of $v$in T_j, which represents the purchase quantity of v.

For instance, consider the transaction database D of Table 1, which will be used throughout the paper to illustrate the definitions. In that database, items are represented by lower-case letters. The database has eight transactions $(T_1, T_2, ..., T_8)$. The transaction T_3 indicates that items a, b, c, d and e have internal utility values of $1, 5, 1, 3$ and 1, respectively. The external utility values of these items are $5, 3, 1, 2, 3$ and 1, respectively, as presented in Table 2.

Definition 2. ***(Taxonomy)*** *[27,28].* A taxonomy τ is a tree (or a directed acyclic graph) defined for a quantitative transaction database $\mathscr{D}$. It has a leaf for each item $v \in \mathscr{I}$. Each inner node is an abstract category that aggregates all descendant leaf nodes (items) or descendant categories into a higher-level category. A child-parent edge between two (generalized) items v, w in τ represents an "is-a" relationship. Inner nodes are called generalized items. The set containing all these generalized items is denoted as GI, and the set containing both generalized and leaf items is denoted as AI. Hence, $AI = GI \cup \mathscr{I}$. Also, let there be a relation $LR \subseteq GI \times \mathscr{I}$ such that $(g, v) \in LR$ if there is a path from g to v. And, let there be a relation $GR \subseteq AI \times AI$ such that $(d, f) \in GR$ if there is a path from item $d$to item f. An itemset $P (P \subseteq AI)$is a set of items, such that and $\nexists v, w \in P | v \in Desc(w, \tau)$. An itemset P is a generalized itemset if $\exists g \in P$ where $g \in GI$. The definition of the concept of descendants of a node w in a taxonomy τ, denoted as $Desc(w, \tau)$, is given in Definition 3.

Definition 3. ***(Descendant)*** *[28].* The leaf items of a generalized item g in a taxonomy τ are all leaves that can be reached by following paths starting from g. This set is formally defined as $Leaf(g, \tau) = \{v | (g, v) \in LR\}$. The descendant items of a (generalized) item d is the set $Desc(d) = \{f | (d, f) \in GR\}$. The level $level(d)$ denotes the number of edges to be traversed to reach an item d starting from the root node of τ.

For example, in the taxonomy shown in Fig. 2, we have $Leaf(\{X\}, \tau) = \{a, b, c\}$ and $Desc(\{X\}) = \{Y, a, b, c\}$, and $level(Y) = 2$.

Definition 4. ***(Utility of an item/itemset)*** *[28].* An item v in a transaction T_j has a utility value, denoted as $u(v, T_j)$, which is calculated as $q(v, T_j) \times p(v)$, in which $q(v, T_j)$ is the internal utility of v in T_j. An itemset $P (X \subseteq I)$ in a transaction $T_j$has a utility value calculated as $u(P, T_j) = \sum_{v \in P} u(v, T_j)$. Let $g(P)$ be the set of transactions containing P in $\mathscr{D}$. The itemset P has a utility value in the database $\mathscr{D}$, which is calculated as $u(P) = \sum_{T_c \in g(P)} u(v, T_j)$.

Table 1
An example quantitative transaction database.

TID	Transactions	TU
T_1	(a, 1), (c, 1), (d, 1)	8
T_2	(a, 2), (c, 6), (e, 1)	19
T_3	(a, 1), (b, 5), (c, 1), (d, 3), (e, 1)	30
T_4	(b, 4), (d, 3), (e, 1)	21
T_5	(a, 1), (b, 1), (c, 1)	9
T_6	(d, 2), (e, 4), (f, 2)	18
T_7	(b, 2), (c, 2)	8
T_8	(c, 3), (d, 2), (e, 3)	16

Table 2
Unit profits of all items in $\mathscr{I}$.

Item	Unit profit
a	5
b	3
c	1
d	2
e	3
f	1

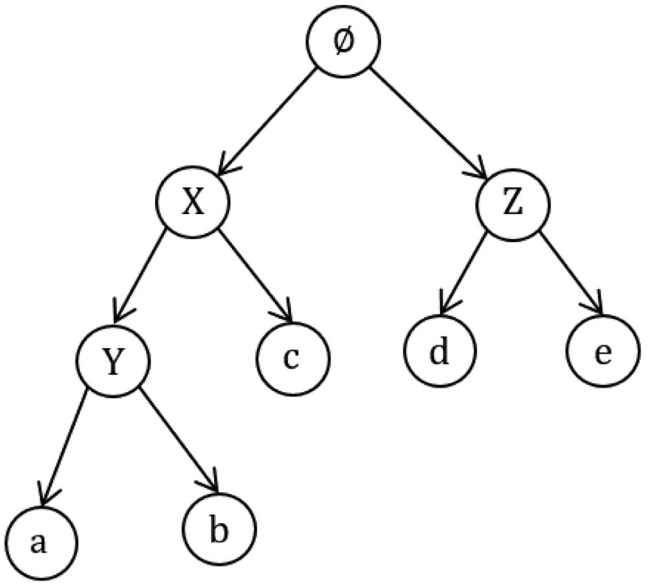

Fig. 2. A taxonomy of items.

For instance, the utility of a in T_3 is $u(\{a\},T_3)=5\times1=5$. The utility of itemset $P=\{a,c\}$ in $T_3$is $u(\{a,c\},T_3)=u(\{a\},T_3)+u(\{c\},T_3)=5\times1+1\times1=6$. The utility of $P=\{a,c\}$ in $\mathscr{D}$ (the whole Table 1) is $u(\{a,c\})=u(\{a\})+u(\{c\})=u(\{a\},T_1)+u(\{a\},T_2)+u(\{a\},T_3)+u(\{c\},T_1)+u(\{c\},T_2)+u(\{c\},T_3)=28$.

Definition 5. ***(Utility of a generalized item/itemset)*** *[28].* A generalized item g in a transaction T_j has its utility, denoted as $u(g,T_j)$, calculated as $u(g,T_j)=\sum_{v\in Leaf(g,\tau)}p(v)\times q(v,T_j),\forall v\in AI$. The utility of a generalized itemset GP in a transaction T_j is determined as $u(GP,T_j)=\sum_{d\in GP}u(d,T_j)$. The utility of a generalized itemset GP in $\mathscr{D}$ is calculated as $u(GP)=\sum_{T_j\in g(GP)}u(GP,T_j)$, in which $g(GP)$is defined as follows: $g(GP)=\{T_j\in\mathscr{D}|\exists P\subseteq T_j\wedge PisaleafofGP\}$.

For instance, the generalized item Y in T_3 has a utility of $u(Y,T_3)=u(a,T_3)+u(b,T_3)=1\times5+5\times3=20$. The descendant of the generalized itemset $P=\{Y,c\}$ in T_3 has a utility of $u(\{Y,c\},T_3)=u(\{Y\},T_3)+u(\{c\},T_3)=20+1\times1=21$. Moreover, the generalized itemset $P=\{Y,c\}$ has a utility of $u(\{Y,c\})=u(\{Y,c\},T_1)+u(\{Y,c\},T_2)+u(\{Y,c\},T_3)+u(\{Y,c\},T_5)+u(\{Y,c\},T_7)=60$.

Based on the above definitions and a user-specified minimum utility threshold μ, an itemset X is considered a cross-level high-utility itemset (CLHUI) if it is a high-utility itemset, which satisfies the μ threshold, and contains items from different abstraction levels.

Definition 6. ***(Cross-level high-utility itemset mining)***. The task of mining cross-level high-utility itemsets (CLHUIMs) is to discover the set of all cross-level HUIs (CLHUIs). A (generalized) itemset is called a CLHUI if and only if its utility is no less than a user-specified minimum utility threshold μ: $u(P)\geq\mu$ [28].

4. Proposed algorithm

This section introduces the proposed FEACP algorithm, which is designed to efficiently enumerate all MLHUIs in a quantitative database $\mathscr{D}$ according to a taxonomy τ and the μ threshold. It is assumed without loss of generality that a total order $\prec$ is defined on items as follows: $a\prec b$ for two items $a,b\in\mathscr{I}\cup GI$, if $level(a)<level(b)$ or $level(a)=level(b)\wedge GTWU(a)>GTWU(b)$.This ordering between levels ensures that the algorithm considers generalized items before their descendant items, which is important as it enables itemset pruning in the search space. This ordering is defined based on the GTWU measure as follows.

Definition 7. ***(Generalized Transaction Weighted Utilization – GTWU)*** *[28].* A transaction T_c has its transaction utility calculated as $TU(T_j)=\sum_{x\in T_j}u(x,T_j)$. For an itemset $P\subseteq I$, the $GTWU$ is computed as $GTWU(P)=\sum_{T_j\in g(P)}TU(T_c)$. In the case of $P\subseteq GI$, the $GTWU$ is obtained as $GTWU(GP)=\sum_{T_j\in g(i\in Leaf(GP,\tau))}TU(T_j)$.

For example, the *TU* values of transactions T_1 to T_8 for the running example of Table 1 are $8, 19, 30, 21, 9, 18, 8$ and 16, respectively. Besides, $GTWU(\{a\}) = TU(T_1) + TU(T_2) + TU(T_3) + TU(T_5) = 8 + 19 + 30 + 9 = 66$. $GTWU(\{Y\}) = TU(T_1) + TU(T_2) + TU(T_3) + TU(T_4) + TU(T_5) + TU(T_7) = 8 + 19 + 30 + 21 + 9 + 8 = 95$.

For any itemset X, it can be seen that $GTWU(P) \geq u(N), \forall N \supseteq P$ (the *GTWU* of P is a utility-based upper bound on P and all its supersets). To eliminate unpromising itemsets from the search space, the following GTWU-based pruning property is used.

Property 1. ***(GTWU-based pruning)*** *[28].* For any itemset P, P and all its supersets are low utility itemsets if $GTWU(P) < \mu$.

The proposed algorithm explores the search space of itemsets using a depth-first search, which is based on a fixed processing order on items from I called $\succ$. The algorithm checks conditions to prune unpromising itemsets early. The search is conducted by starting from higher abstraction levels then moving towards lower abstraction levels. The search space of all itemsets is of size $2^{|I|}$, which can be represented as a set-enumeration tree. The algorithm explores this search space using a depth-first search starting from the root (the empty set). For any itemset, the algorithm recursively appends one item at a time to this itemset according to the order, to generate larger itemsets (extensions).

Definition 8. ***(Extension)*** *[28].* An extension of an itemset P is an itemset obtained by adding an item v to P. The set of all extensions of P is $E(P)$, $E(P) = \{v | v \in AI \wedge v \succ w, \forall w \in P, v \notin Desc(w, \tau)\}$.

To allow the quick calculation of the utility of an itemset and its extensions, the definition of utility-list is needed. Based on the taxonomy defined in *Definition 2*, *Definition 10* presents the utility list structure for each itemset considered during the mining process. The utility-list structure was previously used in several HUIM algorithms to allow fast utility computation. It stores utility information as well as a concept of remaining utility that is useful for reducing the search space.

Definition 9. ***(Remaining utility)***. The remaining utility of an itemset P in a transaction T_j is defined as $re(P, T_j) = \sum_{v \in T_j \wedge v \in E(P)} u(v, T_j)$ [12–15].

Definition 10. *(Utility-list).* A utility-list is a set of tuples such that for each transaction T_j in $\mathscr{D}$ where a (generalized) itemset P appears, there exists a tuple $(j, iutil, rutil)$, where $iutil = u(P, T_j)$ and $rutil = re(P, T_j)$ [12–15].

For example, if the total order is $X \prec Z \prec Y \prec c \prec d \prec e \prec g \prec a \prec b$, then the itemset $\{X, d\}$ has the utility-list: $\{(1, 8, 0), (3, 27, 3), (4, 18, 3)\}$.

The CLH-Miner algorithm builds a tax-utility-list for each promising item and builds a tax-utility-list for each (generalized) itemset. The major difference between the tax-utility-list of CLH-Miner and the utility list in this paper is that the tax-utility-list contains children, which is a set of pointers to the tax-utility-list of the child items of the itemset in the taxonomy. The algorithm in this paper uses the utility list structure to keep track of the utility value of items in transaction, so the children's pointers are unnecessary. Furthermore, the difference between the utility list presented in this paper and the traditional utility list [12,15] is that the field $rutil = re(P, T_j)$ stores the remaining utility of itemset P in transaction T_j, and the items appearing after P in T_j that have no hierarchical relationship to P and themselves.

The proposed algorithm scans the database one transaction at a time to create a hash-table containing tuples of the form $\langle Item, iutil \rangle$. When scanning a transaction T_j, for each item v of that transaction its generalized item P is updated in the hash-table. If P is not already in the hash-table, then a new tuple is constructed as $\langle X, u(v, T_j) \rangle$. Otherwise, the value of item P is updated as $iutil = iutil + u(v, T_j)$. After the database scan is finished, the *rutil*value of each generalized item is calculated by scanning each transaction T_j again to create a tuple $\langle j, iutil, rutil \rangle$ and obtain each generalized item's utility-list. These processes speed up the utility calculations for generalized items.

Table 3
The hash-table obtained from transaction T_1 after reading each item.

Hash-table	Item
Item*Iutil*	a
Y5	
X5	
Item*Iutil*	c
Y5	
X6	
Item*Iutil*	d
Y6	
X5	
Z2	

For instance, the hash-table updated after reading transaction T_1 from the database $\mathscr{D}: \{(a,1),(c,1),(d,1)\}$ is presented in Table 3.

The CLH-Miner algorithm builds a tax-utility-list for each promising item and builds a tax-utility-list for each (generalized) itemset P visited in the search space by joining the tax-utility-list of items in $Leaf(P,\tau)$ with those of generalized items. In the worst case this algorithm requires $O(n \times m \times h)$ space to join m tax-utility-lists of items in $Leaf(P,\tau)$, whereas n denotes the number of transactions and $h = |GI|$.

To effectively reduce the search space when mining cross-level itemsets, the following upper bound is needed to prune an unpromising item and all its extensions.

Definition 11. ***(Upper bound on the utility using the remaining utility)*** *[13,28]*). The remaining utility upper bound of an itemset is calculated as $reu(P) = u(P) + re(P)$. This value can be quickly determined by scanning through the utility-list of P by adding all *iutil* and *rutil* values.

Property 2. *(Pruning using the remaining utility upper bound).* Itemset P and all its extensions are called low utility itemsets if $reu(P) < \mu$, and they can be safely removed from the search space.

This property was introduced in [28], but a proof was not given. Thus, in this paper, this property is proved as follows.

Proof:. *Let P' be an extension itemset of P and for each transaction $\supseteq P' \supset P$.*

In the case where P is not a generalized itemset, the property was proved in [16].

Otherwise, P is a generalized itemset.

Let P' be an extension itemset of $P \Rightarrow (P' - P) = P'/P$

$$P \subset P' \subseteq t \Rightarrow \left(P'/P\right) \subseteq (t/P)$$

$$u\left(P', t\right) = u(P,t) + u\left(\left(P'/P\right), t\right) = u(P,t) + \sum\nolimits_{i \in T_j \bigwedge i \in leaf\left(\left(P'/P\right),\tau\right)} u(i,t)$$

$$u(P,t) + \sum\nolimits_{i \in T_j \bigwedge i \in leaf\left(\left(P'/P\right),\tau\right) u(i,t) \le u(P,t) + \sum_{i \in T_j \bigwedge i \in E(P)} u(i,t) = u(P,t) + re(P,t)}$$

Suppose $g(P)$ denotes the set of transactions containing P, and $g(P')$ those of P', then:

$$P \subset P' \Rightarrow g\left(P'\right) \subseteq g(P)$$

$$u\left(P'\right) = \sum\nolimits_{t \in g\left(P'\right)} u\left(P', t\right) \le \sum\nolimits_{t \in g\left(X'\right)} u(P,t) + re(P,t) \le \sum\nolimits_{t \in g(P)} u(P,t) + re(P,t) = reu(P)$$

So, if $reu(P) < \mu$, itemset P and all of its extensions are low utility itemsets.

To reduce the database size and cost of reading the database, the high-utility projection database (HDP) strategy is used.

Definition 12. *(Projected database). The set of items obtained by projecting a transaction T_j over an itemset P is denoted as $(T_j)_P$ and calculated as $(T_j)_P = \{k | k \in T \wedge k \in E(P)\}$. The multiset obtained after projecting itemset P over all transactions in $\mathscr{D}$, denoted $\mathscr{D}_P$, is calculated as $\mathscr{D}_P = \{(T_j)_P | T_j \in \mathscr{D} \wedge (T_j)_P \neq \varnothing\}$.*

For instance, for the database $\mathscr{D}$ given in Table 1 and $P = \{b\}$, the database $\mathscr{D}_P$ can be constructed, which contains the following transactions: $(T_3)_P = \{c,d,e\}$, $(T_4)_P = \{d,e\}$ and $(T_7)_P = \{c\}$.

To reduce the search space, the following tighter, more effective upper bounds on the utility are utilized. They are the local utility and sub-tree utility, which were introduced in previous state-of-the-art algorithms [13,14,16], but are here adapted for taking a taxonomy into account with generalized items.

Definition 13. ***(Local utility)****. Let P be an itemset. Consider an item $v \in E(P)$. The local utility of v w.r.t P, denoted as $lu(P, v)$, is defined as follows.*

In the case where v is a specialized item, the local utility value of v w.r.t itemset P is computed using the formula introduced in [13,14,16]:

$$lu(P, v) = \sum\nolimits_{T_j \in g(P \cup \{v\})} \left[u\left(P, T_j\right) + re\left(P, T_j\right)\right]$$

If item v is a generalized item in taxonomy, the local utility of v w.r.t itemset P is calculated as follows.

$$lu(P, v) = \sum\nolimits_{T_j \in g(P \cup i \in \{Leaf(v,\tau)\})} \left[u\left(P, T_j\right) + re\left(P, T_j\right)\right]$$

The local utility is an upper bound used to prune the search space using the following property:

Property 3. *(Local utility as an upper bound). Let M be an extension of an item v such that $v \in M$. Then, $lu(M, v) > u(M)$.*

Proof:. *Let $N = M \cup \{v\}$. If v is a specialized item, the property is proved in [16]. If v is instead a generalized item, $u(M) = \sum_{T_j \in g(M \cup i \in \{Leaf(v,\tau)\})} \left[u(M, T) + \sum_{i \in Leaf(v,\tau)} u(i, T)\right]$. In that case, the local utility of P w.r.t to v is $lu(P, v) = \sum_{T_j \in g(P \cup i \in \{Leaf(v,\tau)\})} \left[u(P, T) + \sum_{i \in Leaf(v,\tau)} u(i, T_j) + u((M\P)(v, \tau))\right] \geq u(M)$. Thus $lu(P, v) \geq u(P)$.*

Consider the running example of Table 1, the local utility value of itemset$\{a\}$ with $P = \{\varnothing\}$ is $lu(\varnothing, \mathrm{a}) = 68$, which is given in Table 4.

The local utility of an itemset is always no less than its sub-tree utility. Hence, an itemset $P \cup \{v\}$ and all its extensions are low utility itemsets if $lu(P, v) < \mu$, which means that an item v can be safely ignored from all sub-trees of P.

Definition 14. *(Sub-tree utility). Let P be an itemset and v be an item that can be used to extend P based on the depth-first search (i.e., $v \in E(P)$).*

In the case where v is a specialized item, the sub-tree utility value of v w.r.t itemset P is computed by the formula introduced in [13,14,16]:

$$su(P, v) = \sum_{T_j \in g(P \cup \{v\})} \left[u(P, T_j) + u(v, T_j) + \sum_{i \in T_j \wedge i \in E(P \cup \{v\})} u(i, T_j)\right]$$

Calculating the sub-tree utility value of a generalized item with respect to an itemset is one of the important contributions of this paper. If item v is a generalized item in taxonomy, the sub-tree utility of v w.r.t itemset X is calculated as follows.

$$su(P, v) = \sum_{T_j \in g(P \cup i \in \{Leaf(v,\tau)\})} \left[u(P, T_j) + \sum_{i \in Leaf(v,\tau)} u(i, T) + \sum_{i \in T_j \wedge i \in E(P \cup \{v\})} u(i, T_j)\right]$$

The sub-tree utility is used as an upper bound for pruning the search space using the following property:

Property 4. *(Sub-tree utility as an upper bound). Let there be an itemset α, an item $v \in E(P)$, and an itemset $N = P \cup \{v\}$ where P is an extension of P, $su(P, v) \geq u(N)$.*

Proof:. *Let $N = P \cup \{v\}$. If v is a specialized item, the property is proved in [16]. For the other case where v is a generalized item, $u(N) = \sum_{T_j \in g(P \cup i \in \{Leaf(v,\tau)\})} \left[u(P, T_j) + \sum_{i \in Leaf(v,\tau)} u(i, T_j)\right]$. The sub-tree utility of P w.r.t v is $su(P, v) = \sum_{T_j \in g(P \cup \{Leaf(v,\tau)\})} \left[u(P, T_j) + \sum_{i \in Leaf(v,\tau)} u(i, T_j) + \sum_{i \in T_j \wedge i \in E(P \cup \{v\})} u(i, T_j)\right] = \sum_{T_j \in g(P \cup i \in \{Leaf(v,\tau)\})} \left[u(N, T_j) + \sum_{i \in T_j \wedge i \in E(P \cup \{v\})} u(i, T_j)\right] \geq u(N)$. Thus $su(P, v) \geq u(N)$. Let's consider another itemset M that is an extension of N with a generalized item v'. Then, $u(M) = \sum_{T_j \in g(N \cup i \in Leaf(v',\tau))} u(N, T_j) + \sum_{T_j \in g(N \cup i \in Leaf(v',\tau))} \sum_{i \in Leaf(v',\tau)} u(i, T_j)$. Because $g(M) \subseteq g(N)$ and $E(v') \subseteq E(v)$, we have $su(P, v) \geq u(M)$.*

It can be demonstrated that the itemset $P \cup \{v\}$ and all its extensions are low-utility itemsets if $su(P, v) < \mu$, and they can thus be discarded from the search space [13,14,16]. The proposed algorithm utilizes this property for search space pruning.

Consider the running example of Table 1 and $P = \{\mathrm{a}\}$ as an example. We have that $su(P, \mathrm{c}) = (5 + 1 + 2) + (10 + 6 + 3) + (5 + 1 + 9) + (5 + 1 + 0) = 48$.

Property 5. *(Search space pruning using upper bounds). Let Pbe an itemset, a generalized item v, and an itemset $N = P \cup \{v\}$. Then, $GTWU(N) \geq lu(P, v) \geq reu(N) = su(P, v)$.*

Table 4
Local utility of items in the running example.

Item	a	b	c	d	e	f	X	Y	Z
lu	68	59	92	93	103	17	113	95	112

Proof:. *If v is a specialized item, the property was proved in [16]. Otherwise, v is a generalized item where* $GTWU(N) = \sum_{T_j \in g(P \cup \{i \in Leaf(v,\tau)\})} TU(T_j)$, $lu(P,v) = \sum_{T_j \in g(P \cup i \in \{i \in Leaf(v,\tau)\})} [u(P,T_j) + re(P,T_j)]$ *and* $u(P,T_j) + re(P,T_j) \leq TU(T_j)$. *Thus* $GWU(N) \geq lu(P,v)$. $reu(N) = \sum_{T_j \in g(p \cup \{i \in Leaf(v,\tau)\})} [u(N,T_j) + re(P \cup \{v\}, T_j)]$ *and* $re(P,T_j) \geq re(P \cup \{v\}, T_j)$ *so* $lu(P,v) \geq reu(N)$. *Finally,* $su(P,v) = \sum_{T_j \in g(P \cup i \in \{Leaf(v,\tau)\})} \left[u(P,T_j) + \sum_{i \in Leaf(v,\tau)} u(i,T_j) + \sum_{i \in T_j \wedge i \in E(P \cup \{v\})} u(i,T_j)\right]$ $= \sum_{T_j \in g(P \cup i \in \{Leaf(v,\tau)\})} [u(N,T_j) + \sum_{i \in T_j \wedge i \in E(P \cup \{v\})} u(i,T_j)]$. *Because* $re(P \cup \{v\}, T_j) = \sum_{i \in T_j \wedge i \in E(P \cup \{v\})} u(i,T_j)$, *it follows that* $reu(N) = su(P,v)$.

According to *Property 5*, the local utility upper bound is tighter than the *GTWU*. Thus, the local utility can be more effective for pruning the search space.

Regarding the sub-tree utility, if $su(P,v) < \mu$, the algorithm will prune both v and the descendant nodes of v in the depth first search tree. While with $N = P \cup \{v\}$,$reu(N)$ only prunes the descendant nodes of v. Thus, the sub-tree utility is a tighter upper bound than the $reu(N)$.

Based on the local utility and sub-tree utility, the following definitions present the concepts of primary and secondary items, which means items that need to be considered to explore the search space. These concepts [16] are extended for generalized items to mine cross-level itemsets.

Definition 15. ***(Primary items)****. Let P be an itemset. The set containing all primary items of P, denoted as* $\mathbb{P}(P)$, *is defined as* $\mathbb{P}(P) = \{v | v \in E(P) \wedge su(P,v) \geq \mu\}$.

Definition 16. ***(Secondary items)****. Similarly, let P be an itemset. Based on the local utility, the set containing all secondary items of P, denoted as* $\mathbb{S}(P)$, *is determined as* $\mathbb{S}(P) = \{v | v \in E(P) \wedge lu(P,v) \geq \mu\}$.

The uses of Primary items and Secondary items is similar to in EFIM and iMEFIM [16]. However, the sets $\mathbb{P}(P)$ and $\mathbb{S}(P)$ in the current work can contain both generalized and specialized items that have no relationship to each item in itemset P.

Because $lu(P,v) \geq su(P,v)$, it follows that $\mathbb{P}(P) \subseteq \mathbb{S}(P)$ (*Property 5*).

Definition 17. *(Utility-Bin array). Consider the set $\mathscr{I} \cup GI$ containing all distinct items in $\mathscr{D}$ and generalized items in GI. The utility-bin array Uis an array of length $|\mathscr{I} \cup GI|$ that has an entry $U[v]$ for each item $v \in \mathscr{I} \cup GI$ called a utility-bin. Each utility-bin can be used to hold a utility value and is initialized to zero.*

An algorithm can utilize a utility-bin array to determine the GTWU of any item when scanning item v in each transaction T_j, $GTWU[v] = GTWU[v] + TU(T_j)$.For $v \in T \cap E(P)$, the utility-bin is used to calculate the sub-tree utility w.r.t. an itemset X as$su[v] = su[v] + u(X,T_j) + u(v,T_j) + \sum_{i \in T_j \wedge i \succ v} u(i,T_j)$.It also supports the calculation of the local utility w.r.t. an itemset X as $lu[v] = lu[v] + u(P,T_j) + re(P,T_j)$.

The Utility-Bin array is defined in [16], but in this paper we extend this array to store and calculate the value of the sub-tree utility, *GTWU* and local utility of generalized items in the set *GI*. When scanning the database to calculate the upper bound, the Utility-Bin array regards generalized items as specialized items.

In traditional HUIM algorithms, the upper bounds usually contain only the utility of the promising items. In CLHUI mining, if the upper bounds only contain the utility of the promising specialized items, they will miss out several generalized items. This is because the utility of these generalized items is aggregated from the unpromising leaf items, which were previously discarded. Therefore, to accurately calculate these upper bounds we revise the remaining utility and sub-tree utility of an itemset using the following definitions.

Definition 18. ***(The prefix of an itemset)****. Let P be an itemset. The prefix of an itemset is an itemset, denoted as* $pe(P)$, *obtained by removing the last item of P, denoted as* $li(P)$.

For example: Let an itemset $K = \{ab\}$, $pe(K) = \{a\}, li(K) = \{b\}$.

To compute the remaining utility and sub-tree utility of an itemset P, *Definition 9* and *Definition 14* are used, respectively. However, that would require scanning from the start of a transaction containing P. Thus, the following properties are proposed to allow the rapid calculation of the remaining utility and sub-tree utility.

Property 6. ***(Fast computation of remaining utility)****. Let X be an itemset, and the remaining utility X can be computed as*:

$$re(P,T_j) = re(pe(P),T_j) - u(li(P),T_j) - \sum\nolimits_{v \in T_j \wedge v \in E'(P)} u(v,T_j)$$

Whereas

$$E'(P) = \{v | \wedge\, v \in E(pe(P)) \wedge v \prec li(P) \wedge v \notin Desc(li(P),\tau)\}$$

And $re(\varnothing, T_j) = TU(T_j)$.

Proof:. $re(pe(P),T_j) = \sum_{v\in T_j \wedge v\in E(pe(P))} u(v,T_j) = u(li(P),T_j) + \sum_{v\in T_j \wedge v\in E(pe(P)) \wedge v\prec li(P) \wedge v\notin Desc(li(P),\tau)} u(v,T_j) + \sum_{v\in T_j \wedge v\in E(pe(P)) \wedge v\succ li(P) \wedge v\notin Desc(li(P),\tau)} u(v,T_j) = u(li(P),T_j) + \sum_{v\in T_j \wedge v\in E(pe(P)) \wedge v\prec li(P) \wedge v\notin Desc(li(P),\tau)} u(v,T_j) + re(P,T_j)$.

Consider the running example of Table 1 and $P = \{a,d\}$. We have that $re(P,T_3) = re(a,T_3) - (u(b,T_3) + u(c,T_3) - u(d,T_3)) = re(\varnothing,T_3) - u(a,T_3) - (u(b,T_3) + u(c,T_3) - u(d,T_3)) = TU(T_3) - u(a,T_3) - (u(b,T_3) + u(c,T_3) - u(d,T_3)) = 30 - 5 - 15 - 1 - 2 = 3$.

Property 7. ***(Fast computation of sub-tree utility)***. *Let P be an itemset. The sub-tree utility of an itemset P w.r.t item v can be computed as*:

$$su(P,v) = \sum_{T_j\in g(P\cup i\in\{Leaf(v,\tau)\})} \left[u(P,T) + \sum_{i\in T_j\wedge i\in E(P)} u(i,T_j) - \sum_{i\in T_j\wedge i\in E(P\cup\{v\})\wedge i\prec v} u(i,T_j)\right]$$

Proof:. $su(P,v) = \sum_{T_j\in g(P\cup i\in\{Leaf(v,\tau)\})} \left[u(P,T_j) + \sum_{i\in Leaf(v,\tau)} u(i,T_j) + \sum_{i\in T_j\wedge i\in E(P\cup\{v\})} u(i,T_j)\right] = \sum_{T_j\in g(P\cup i\in\{Leaf(v,\tau)\})} [u(pe(P),T_j) + u(li(P),T_j) + \sum_{i\in Leaf(v,\tau)} u(i,T_j) + \sum_{i\in T_j\wedge i\in E(P\cup\{v\})} u(i,T_j) + \sum_{i\in T_j\wedge i\in E(P\cup\{v\})\wedge i\prec v} u(i,T_j) - \sum_{i\in T_j\wedge i\in E(P\cup\{v\})\wedge i\prec v} u(i,T_j)]$

And:

$$\sum_{i\in Leaf(v,\tau)} u(i,T_j) + \sum_{i\in T_j\wedge i\in E(P\cup\{v\})} u(i,T_j) + \sum_{i\in T_j\wedge i\in E(P\cup\{v\})\wedge i\prec v} u(i,T_j) = \sum_{i\in T_j\wedge i\in E(pe(P)\cup li(P))} u(i,T_j)$$

$$su(P,v) = \sum_{T_j\in g(P\cup i\in\{Leaf(v,\tau)\})} \left[u(P,T_j) + \sum_{i\in T_j\wedge i\in E(P)} u(i,T_j) - \sum_{i\in T_j\wedge i\in E(P\cup\{v\})\wedge i\prec v} u(i,T_j)\right]$$

Property 6 and Property 7 were introduced and proven to accurately calculate the upper bounds. These values can be calculated as follows:

- The remaining utility: assuming a transaction $(T_j)_P$ is obtained after projecting T_j on an itemset P, the remaining utility of P is computed and saved to a variable. This value will be used to calculate the remaining utility of supersets of P in the transaction. $(T_j)_P$is then scanned from left to right. For each scanned item, the utility of that item is subtracted from the remaining utility of the previous item to obtain the remaining utility.
- The process of calculating the subtree-utility of an itemset w.r.t an item v is similar to that of the remaining utility. The projected transaction $(T_j)_P$ stores the utility and remaining utility of P. $(T_j)_P$ is then scanned from left to right to remove the utility of the item before v from the remaining utility.

4.1. FEACP algorithm

The proposed FEACP algorithm is presented in Algorithm 1. It accepts three input parameters that are a quantitative database $\mathscr{D}$, a taxonomy τ, and the user-defined threshold μ. The overall process is done as follows:

- Line #1 initializes the current itemset P to the empty set ($\varnothing$). In the case where $P = \varnothing$, $lu(P,v) = GTWU(v)$.
- Line #2 scans the database $\mathscr{D}$ along with the taxonomy τ to compute the local utility of every item $v \in \mathscr{I}$ and the *GTWU*s of generalized items of v using a utility-bin array. This phase utilizes *Definition 13*.
- In the next line (Line #3), the algorithm compares the local utility of each item to μ to construct the set of all secondary items of P, $\mathbb{S}(P)$, using *Property 3* and *Definition 16*.
- Then at line #4, $\mathbb{S}(P)$ is sorted based on the total order $\prec$ of levelsand the *TWU* .
- At line #5, the algorithm then performs another database scan of $\mathscr{D}$ using taxonomy τ to find all generalized items corresponding to secondary items in each transaction, and obtains their *Tid*, utility and remaining utility. For each transaction, a tuple $\langle j, iutil, rutil\rangle$ in the utility list of this generalized item is created to store all the obtained information, which is based on *Definition 10*. All items that are not in $\mathbb{S}(P)$ are removed from the transaction to reduce memory usage. The algorithm then reduces the size of the database $\mathscr{D}$ by removing all transactions that become empty after removing these items.
- Line #6 performs a third scan of the database and taxonomy to compute the sub-tree utility of every item in $\mathbb{S}(P)$ using *Definition 14*.
- Based on *Definition 15*, line #7 constructs the primary set with respect to itemset P, $\mathbb{P}(P)$. These two constructed sets, $\mathbb{P}(P)$ and $\mathbb{S}(P)$, are used to reduce the search space by pruning all itemsets and their extensions that do not satisfy the μ threshold using *Property 3* and *Property 4* of the local utility and sub-tree utility, respectively.
- With all the necessary information gathered, the DFS-based ***Search*** procedure (Algorithm 2) is executed at line #8 to traverse the search space recursively and extend the initial itemset P to find all CLHUIs.

Algorithm 1: The **FEACP** Algorithm

Input: quantitative database $\mathscr{D}$, taxonomy τ, thresholdμ
Output: all discovered CLHUIs
1. $P = \varnothing$;
2. Scan $\mathscr{D}$, τ and use a utility-bin array to calculate $lu(P, v)$ for each $v \in \mathscr{I}$, and each generalized item $g \in GI$;
3. $\mathbb{S}(P) = \{v | v \in \mathscr{I} \wedge lu(P, v) > \mu \wedge g | g \in GI \wedge lu(P, g) > \mu\}$;
4. Calculate the total $\prec$ order of *Level* and of *GTWU* values on $\mathbb{S}(P)$;
5. Scan $\mathscr{D}$ to store each generalized item $g \in \mathbb{S}(P)$ in each transaction, discard every item $v \notin \mathbb{S}(P)$ from transactions, sort items in each transaction, delete empty transactions, and then build and store the utility-list of each generalized item of the database in a variable *UtilityList*;
6. Using a utility-bin array, compute the sub-tree utility $su(P, v)$ of each item v and $g \in \mathbb{S}(P)$ by scanning $\mathscr{D}$;
7. $\mathbb{P}(P) = \{v | v \in \mathbb{S}(P) \wedge su(P, v) \geq \mu \wedge g | g \in \mathbb{S}(P) \wedge su(P, g) \geq \mu\}$;
8. **SEARCH** (P, $\mathscr{D}$, $\mathbb{P}(P)$, $\mathbb{S}(P)$, μ, *UtilityList*);

Algorithm 2: The **SEARCH** procedure

Input: itemset P, P-projected database $\mathscr{D}_P$, primary items of P: $\mathbb{P}(P)$, secondary items of P: $\mathbb{S}(P)$, μ, *UtilityList*
Output: all extensions of P that are CLHUIs.
1. **FOR EACH** item$v \in \mathbb{P}(P)$ **DO**
2. $N = P \cup \{v\}$, $ExtensionUL = \varnothing$, $\mathbb{S}(P)' = \{z \in \mathbb{S}(P) | z \notin Desc(v)\}$;
3. Scan $\mathscr{D}_P$ to determine $u(N)$, construct $\mathscr{D}_N$, remove every $item \in Desc(v)$ from $\mathscr{D}$ and remove empty transactions;
4. **IF** $u(N) \geq \mu$ **THEN** output N;
5. **IF** v is a generalized item **THEN** $ExtensionUL \leftarrow UtilityList(v)$
6. Scan $\mathscr{D}_N$ to compute $su(N, w)$ and $lu(N, w)$ for every item $w \in \mathbb{S}(N)'$ using two utility-bin arrays; If w is a generalized item with a utility-list $u(w)$in$UtilityList(w)$, then add $u(w)$to $ExtensionUL(w)$;
7.$\mathbb{P}(N) = \{w \in \mathbb{S}(P) | su(Y, w) \geq \mu\}$;
8.$\mathbb{S}(N) = \left\{w \in \mathbb{S}(P)' \middle| lu(Y, w) \geq \mu\right\}$;
9. Remove each item $w \in ExtensionUL$suchthat$lu(Y, w) < \mu$;
10. **SEARCH**(N, $\mathscr{D}_N$, $\mathbb{P}(N)$, $\mathbb{S}(N)$, μ, *ExtensionUL*);
11. **END**

Algorithm 2 accepts six arguments as input: an itemset P to be extended, the projected database $\mathscr{D}_P$ of P, the set of primary and secondary items of P, the minimum utility thresholdμ, and the utility-lists *UtilityList*of generalized items in $\mathscr{D}$. The details of the depth-first search process to further explore a given itemset are described as follows.

- Lines #1 to #11 form a loop to examine every single-item extension of P made using each item $v \in \mathbb{P}(P)$ based on *Definition 8*, that is of the form $N = P \cup \{v\}$, which is given in line #2.
- Considering the extension N, line #3 performs a database scan to determine the utility of N and construct N's projected database, namely $\mathscr{D}_N$, using *Definition 12*. D_N is built to reduce the cost of the database scan, especially for those with long transactions.
- Line #4 checks whether N has a utility greater or equal to μ. If so, then N is output as a CLHUI. Otherwise, N is a low utility itemset.
- In this case, $\mathscr{D}_N$ is scanned once more at line #6 to determine the sub-tree and local utility of each item w that could be used to extend N. This step is required to construct $\mathbb{P}(N)$ and $\mathbb{S}(N)$ at lines #7 and #8, respectively.
- Line #9 prunes each item w from the utility-list if the extension of N using w has a utility value that does not satisfy the μ threshold.
- At line #10, the ***Search*** algorithm is then recursively invoked to continue exploring the search space to extend N.

Based on the properties and theorems presented in previous sections, when FEACP terminates all and only the cross-level high-utility itemsets have been output. Since the taxonomy τ is always considered during the calculation of the GTWU values to construct the utility-lists and during the DFS-based search, the discovered itemsets are cross-level HUIs. Furthermore, *Definitions 3* and *9* also ensure this, since at every expansion of an itemset P using $\mathbb{P}(P)$, the taxonomy τ is always considered.

In the next subsection, an example will be presented to illustrate in detail the process of mining CLHUIs using the proposed FEACP algorithm.

4.2. A descriptive example

This subsection offers a detailed walkthrough of how the proposed FEACP algorithm discovers CLHUIs in the running database $\mathscr{D}$ of Table 1 and Table 2, assuming $\mu = 65$.

- **Step 1.** The current itemset P is initialized to $\varnothing$.
- **Step 2.** For every item $v \in AI$, the value of $lu(P, v)$ is determined. The $lu(P, v)$ values are given in Table 5.
- **Step 3.** Using the calculated $lu(P, v)$ values of every item, the set $\mathbb{S}(P)$ is constructed. In this case, $\mathbb{S}(P) = \{a, c, d, e, X, Y, Z\}$.
- **Step 4.** The set $\mathbb{S}(P)$ is then sorted in ascending order of *level* and then descending order of $lu(P, v)$values. Thus, $\mathbb{S}(P) = \{X, Z, c, d, Y, e, a\}$.
- **Step 5.** Every transaction in $\mathscr{D}$ is then sorted by the order of $\mathbb{S}(P)$. The result is the database given in Table 7. The algorithm then constructs the utility-lists of generalized items, which are shown in Table 6. All items not in $\mathbb{S}(P)$ are removed from all transactions since they cannot be part of a HUI.
- **Step 6.** For every secondary item v, $su(P, v)$ is computed for $P = \varnothing$. The results are given in Table 8.
- **Step 7.** Using the values obtained from the previous step, the set $\mathbb{P}(\alpha)$ is constructed as $\mathbb{P}(P) = \{X, Z, c\}$.
- **Step 8.** The algorithm is then recursively invoked to explore larger itemsets. It is invoked with the following parameters: $Search(P = \varnothing, \mathscr{D}, \mathbb{P}(P) = \{X, Z, c\}, \mathbb{S}(P) = \{X, Z, c, d, Y, e, a\}, \mu = 65)$.

- The item X is first processed by the ***Search*** function, $N = P \cup \{X\} = \{X\}$.
- Each item of $Desc(N)$ is removed. The result is $\mathbb{S}(P)' = \{Z, d, e\}$.
- Because T_6does not contain$Leaf(P, \tau)$, the transactions $T_1, T_2, T_3, T_4, T_5, T_7andT_8$ are scanned to calculate $u(N) = 77$, and the projected database $\mathscr{D}_N$ is constructed, which is shown in Table 9.
- Since $u(N)$ satisfies μ, {X}is output as a CLHUI.
- According to Table 9, T_5 and T_7 are removed since they are now empty transactions. The resulting database is shown in Table 10.
- The utility-list $U(P)$ is then also used to calculate the $lu(P, v)$and $su(P, v)$ values of each item $v \in \mathbb{S}(P)$. The result is given in Table 11.
- Given that $\mu = 65$, the constructed sets $\mathbb{P}(N)$ and $\mathbb{S}(N)$ with respect to N are$\mathbb{P}(N) = \{Z, d\}$ and $\mathbb{S}(N) = \{Z, d, e\}$, respectively.
- Then, the Search function is recursively invoked to discover CLHUIs that can be obtained by extending Y using secondary items, using the following parameters $Search(N = \{P\}, \mathscr{D}_N, \mathbb{P}(N) = \{Z, d\}, \mathbb{S}(N) = \{Z, d, e\}, \mu = 65)$.
  - With $N = \{XZ\}$, it is found that $u(N) = 102 > \mu$.Thus, this itemset is a CLHUI. Moreover, D_Y *is* empty.
  - With $N = \{Xd\}$, $u(N) = 83 > \mu$ and thus it is a CLHUI. The results returned when calling the Search function with $N = \{Xd\}$are given in Table 12 to Table 14.Table 13.
  - The Search function is then again recursively called to extend {Xd}in a similar way. The CLHUI {Xde} is found.

At this step, the FEACP algorithm has now finished processing item P. The next item in the set $\mathbb{S}(N)$ is then considered. For $N = \{Z\}$ and $N = \{c\}$, the calculated utility values are $u(\{Z\}) = 54$ and $u(\{c\}) = 19$, respectively. Thus, no other CLHUIs were discovered. The determined values of $N = \{Z\}$ are presented in
Table 15 to Table 17. The calculated values of $N = \{c\}$ are presented in Table 18 to Table 20. All found CLHUIs are listed in Table 21,.Table 16., Table 19.

5. Evaluation studies

To evaluate FEACP's performance, experiments were performed on a computer with an Intel® Core-i5™ processor clocked at 3.1 GHz and 8 GB of RAM, running the Windows 10 operating system. We compared the performance of FEACP in terms of execution time and peak memory usage for mining CLHUIs against CLH-Miner. The ML-HUI Miner algorithm was not included in this test since it only mines multi-level itemsets, rather than cross-level itemsets. Considering each level of the taxonomy separately is easier than the problem solved by FEACP and CLH-Miner and produces fewer itemsets. To have a better understanding of this latter aspect, we also compared the number of itemsets discovered by the proposed algorithm with CLH-Miner and ML-HUI Miner. We also evaluated the effectiveness of the sub-tree utility pruning of the FEACP algorithm by comparing with a version where only the local utility is used to reduce the search space. That version is denoted as FEACP(lu). It is to be noted that we do not compare this with a variant of CLH-Miner named TKC, which was recently designed to mine the top-k cross-level high-utility itemsets [54]. The reason is that TKC was outperformed in terms of execution time by CLH-Miner on all tested databases [54]. The Java programming language (JDK 8.0) was used to implement the

Table 5
The lu(P, v)value of each item $v \in AI$ for $P = \varnothing$.

Item	a	b	c	d	e	f	X	Y	Z
lu	68	59	92	93	103	17	113	95	112

Table 6
Utility-lists of generalized items X, Y and Z.

Utility-list ofX			Utility-list ofY			Utility-list ofZ		
Tid	Util	Rutil	Tid	Util	Rutil	Tid	Util	Rutil
1	6	2	1	5	3	1	2	6
2	16	3	2	10	6	2	3	16
3	21	9	3	20	7	3	9	6
4	12	9	4	12	6	4	9	0
5	11	0	5	5	6	6	18	0
7	8	0	7	6	2	8	13	3
8	3	13						

Table 7
The preprocessed database.

TID	Items
T_1	$(c,1),(d,1),(a,1)$
T_2	$(c,6),(e,1),(a,2)$
T_3	$(c,1),(d,3),(e,1),(a,1)$
T_4	$(d,3),(e,1)$
T_5	$(c,1),(a,1)$
T_6	$(d,2),(e,4)$
T_7	$(c,2)$
T_8	$(c,3),(d,2),(e,3)$

Table 8
The calculated $su(P,v)$ value of each item $v \in AI$ for $P = \varnothing$.

Item	X	Z	c	D	Y	e	a
su	113	85	71	59	67	45	25

Table 9
Projecting $N = \{X\}$ on $\mathscr{D}$.

TID	Items
T_1	$(d,1)$
T_2	$(e,1)$
T_3	$(d,3),(e,1)$
T_4	$(d,3),(e,1)$
T_5	
T_7	
T_8	$(d,2),(e,3)$

Table 10
The obtained database $\mathscr{D}$ after discarding empty transactions.

TID	Items	Generalized Item
T_1	$(d,1)$	Z
T_2	$(e,1)$	Z
T_3	$(d,3),(e,1)$	Z
T_4	$(d,3),(e,1)$	Z
T_8	$(d,2),(e,3)$	Z

Table 11
The calculated values of lu(N, v) and su(N, v) for N = {X}.

Item	Z	d	e
su	94	75	60
lu	94	75	86

Table 12
The projection of N = {Xd} on $\mathscr{D}$.

TID	Items
T_3	(e, 1)
T_4	(e, 1)
T_8	(e, 3)

Table 13
The lu(Y, v)and su(N, v) values forN = {Xd}.

Item	e
su	67
lu	67

Table 14
The primary and secondary items for N = {Xd}.

$\mathbb{P}(N)$	e
$\mathbb{S}(N)$	e

Table 15
Projecting N = {Z} on $\mathscr{D}$.

TID	Items
T_1	(c, 1), (a, 1)
T_2	(c, 6), (a, 2)
T_3	(c, 1), (a, 1)
T_8	(c, 3)

Table 16
The calculated values of lu(N, v), su(N, v) for N = {Z}.

Item	c	Y	a
su	60	62	34
lu	60	73	42

Table 17
Primary and secondary items of N = {Z}.

$\mathbb{P}(N)$	$\varnothing$
$\mathbb{S}(N)$	Y

Table 18
Projecting N = {c} on $\mathscr{D}$.

TID	Items
T_1	(d, 1), (a, 1)
T_2	(e, 1), (a, 2)
T_3	(d, 3), (e, 1), (a, 1)
T_5	(a, 1)
T_8	(d, 2), (e, 3)

Table 19
The calculated values of lu(N, v) and su(N, v) for N = {c}.

Item	d	Y	e	a
su	39	68	40	39
lu	39	76	50	53

Table 20
Primary and secondary items of N = {Z}.

$\mathbb{P}(N)$	Y
$\mathbb{S}(N)$	Y

Table 21
All found MLHUIs.

CLHUIs	**Utility**
{X}	77
{X, Z}	102
{X, d}	83
{X, d, e}	129

Table 22
Database characteristics.

Database	$\lvert\mathscr{D}\rvert$	$\lvert\mathscr{I}\rvert$	$\lvert GI\rvert$	*MaxLevel*	$\lvert T_{MAX}\rvert$	$\lvert T_{AVG}\rvert$	Density
Foodmart	53,537	1,560	102	5	28	4.60	Sparse
Fruithut	181,970	1,265	43	4	36	3.58	Sparse
Chainstore	1,112,949	40,086	11,936	10	170	7.20	Sparse
Connect	67,557	129	40	4	43	43.00	Dense
Accidents	340,183	468	216	6	51	33.80	Dense
Chess	3,196	75	30	3	37	37.00	Dense

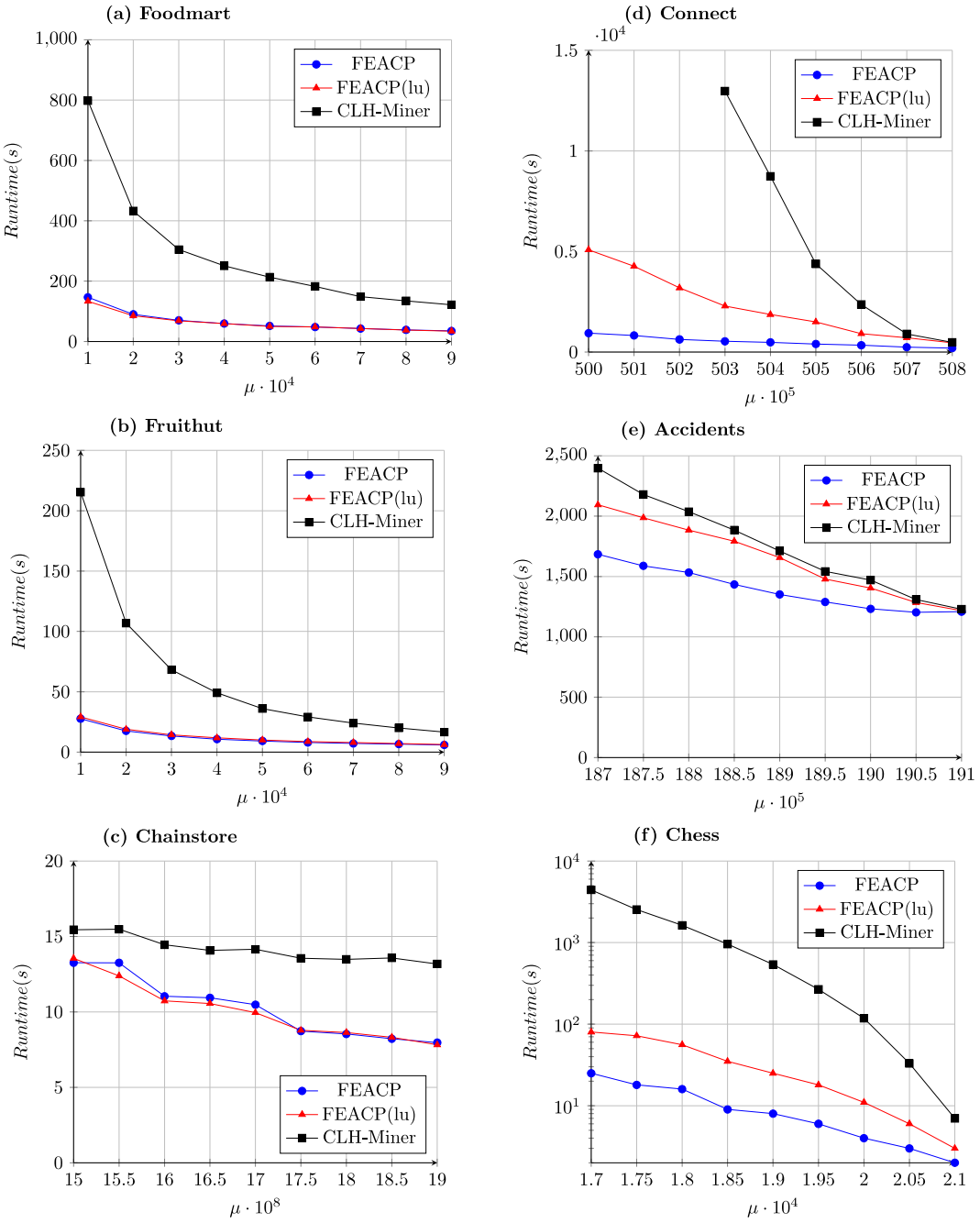

Fig. 3. Runtime comparison on the tested databases.

compared algorithms. Runtime and memory usage were recorded using the Java API. The source code of the FEACP algorithm and all test datasets used in this section are available on GitHub[1].

Six databases with taxonomy were used to evaluate the performance of the tested algorithms. Foodmart[2] and Fruithut[3] contain a built-in taxonomy, while the Chainstore, Connect, Accidents and Chess databases have synthetic taxonomies. Chainstore, Connect, Accidents and Chess were obtained from the SPMF open-source data mining package, while the taxonomy infor-

[1] Source: https://github.com/nguyenthanhtunghutechsg/FEACP-Evaluation
[2] Source: https://github.com/arunkjn/foodmart-mysql
[3] Source: https://data.world/agriculture/fruit-vegetable-market-news

mation of the two first datasets were extracted from the original dataset files, which was obtained from the URL provided in the footnotes.

The characteristics of these databases are listed in Table 22. In that table, $|\mathscr{D}|$ is the transaction count of $\mathscr{D}$, $|\mathscr{I}|$ is the number of distinct items, $|GI|$ is the generalized item count, T_{MAX} is the maximum transaction length, T_{AVG} is the average transaction length, *MaxLevel* is the maximum level in each database and the last column is the density of each database. The last column of the table shows the density of the databases. It can be seen in Table 22 that Chainstore is the largest of all databases, containing over 1 M transactions and 40 K items. Its taxonomy was also synthesized to be the deepest, with 10 levels. Chess is the smallest of all the databases, with over 3 K transactions and 75 items. Its taxonomy is also only three levels. However, by looking at the T_{AVG} and T_{MAX} properties, Chess is the densest database.

First, we evaluated how the runtime of each algorithm is influenced by the μ threshold on the test databases for various μ threshold values. The algorithms were initially executed on the test databases with a high μ threshold and then the threshold was lowered until one of the following conditions was observed: the test algorithms had too long runtimes, threw an out of memory exception, or there was a clear winner. The execution times of the tested algorithms are presented in Fig. 3 for each database. For CLH-Miner, no result is given for the μ threshold from the value of 50.2Mto 50.0M on Connect since the algorithm took over three hours. Hence, FEACP and FEACP(lu) have the best runtimes. FEACP terminated in 628 seconds and FEACP(lu) completed its work in $3,190$ seconds.

The results show that FEACP has much better performance than CLH-Miner for all μ threshold values. The first two databases are Foodmart and Fruithut, which are sparse databases, and the performances of FEACP and FEACP(lu) were about the same, with FEACP(lu) slightly better at low μ thresholds on the Foodmart database. The explanation for this is clearly based on the definitions of sub-tree utility and local utility. Although FEACP can prune a larger number of unpromising candidates using both the sub-tree and local utility upper bounds compared to FEACP(lu), its computational cost is higher than that of FEACP(lu) with only the local utility. For example, on the Foodmart database (Fig. 3a) FEACP finished in 147*s*, FEACP(lu) finished 133*s* and CLH-Miner required 798*s*. The runtime of FEACP on this database is 5.4 times shorter than that of CLH-Miner. The same execution time reduction can be observed on the Fruithut database. Generally speaking, the performances of FEACP and FEACP(lu) are better than those of CLH-Miner, from 3.5 to 5.4 times on the Foodmart database (Fig. 3a) and from 2.8 to 7.8 times on the Fruithut (Fig. 3b) database.

The same results were seen on the Chainstore database (Fig. 3c), which is a large and sparse database with over 1M transactions. The performances of both FEACP and the unoptimized version FEACP(lu) are also better than that of CLH-Miner. However, the speed gained is not as high as on Foodmart and Fruithut, as it is only 1.2 to 1.7 times better. This is due to the nature of a large and sparse database, containing short transactions, combined with 10 abstraction levels and almost 12K generalized items. The algorithms need to consider and prune a large number of cross-level candidates, thus lowering the performance.

The next three databases are Connect, Accidents and Chess. They are all have high density compared to the previous databases. For the Connect database (Fig. 3d), which is a dense database, FEACP has shown its effectiveness with its tight sub-tree utility pruning with an extremely low runtime when compared to FEACP(lu) and CLH-Miner. CLH-Miner's execution time started to exceed 3 h at $\mu = 50.2$M and below, thus no further tests were needed since FEACP took only $628s, 826$s and 937s at the same μ thresholds. The sub-tree utility eliminated numerous itemsets having *su* values that do not meet the μ threshold. While the *GTWU*, which is used in both FEACP(lu) and CLH-Miner, still left behind a lot of candidates to be checked. The FEACP(lu) pruned the candidates using only local utility, and the pruning effectiveness is not as good as that of the sub-tree utility, and its running time increased from 2.3 times to 5.4 times. Furthermore, the runtime of FEACP is over 24 times better than that of CLH-Miner. Thus, the GTWU is clearly not as efficient as the sub-tree utility, which is used by FEACP. The same runtime results are also observed in the Accidents (Fig. 3e) and Chess (Fig. 3f) databases. As observed from the results, FEACP has again shown its efficiency over FEACP(lu) and CLH-Miner. In the Accidents database (Fig. 3e), FEACP is faster than CLH-Miner by from 1.02 times at $\mu = 1.91$M to 1.42 times at $\mu = 1.87$M; it is also faster than the unoptimized version FEACP(lu) by up to 1.2 times. The database that has the highest increase in speed for FEACP is Chess. This is a highly dense database, and so FEACP has the best execution time here (Fig. 3f). It finished the work from 2.0 s to 25.0 s in all the tested μ thresholds, while FEACP(lu) required from 3 to 80 s. CLH-Miner took from 7 s to 4,423 s to finish its task in the same μ range. From the obtained data, it is seen that FEACP is 4 to 177 times faster than CLH-Miner, and 1.5 times to 3.2 times better than its unoptimized version. The vertical axis of the chart in Fig. 3f has to be switched from linear scale into logarithmic to clearly illustrate the large differences in running times of FEACP, FEACP(lu) and CLH-Miner. Thus FEACP, with the efficient pruning strategies applied, can eliminate many candidates found in the dense databases, and significantly reduce the mining time. Based on this evaluation, the FEACP algorithm has the highest mining performance on the dense databases.

The number of discovered itemsets from the test databases is shown in Table 23 and Table 24. Table 23 presents the number of itemsets obtained from the sparse databases, and Table 24 shows the ones obtained from the dense databases. To have a better view about the number of itemsets found by the algorithms, the HUI counts from each database at different μ thresholds are also presented. Since both FEACP and FEACP(lu) return the same number of itemsets, FEACP(lu) is omitted from Table 23 and Table 24. As observed from these tables, FEACP and CLH-Miner return the same set of cross-level HUIs while the number of HUIs returned is much smaller. To obtain the HUIs from the test databases, empty taxonomies were used. The reason for this is that FEACP and CLH-Miner find cross-level itemsets, which contain more combinations of items

Table 23
Number of discovered itemsets in sparse databases at various μ thresholds.

Foodmart			**Fruithut**			**Chainstore**		
μ	FEACP &CLH-Miner	HUIs	μ	FEACP &CLH-Miner	HUIs	μ	FEACP &CLH-Miner	HUIs
10K	2,134,247	1321	10K	23,325	154	1.50B	3	0
20K	599,085	805	20K	6339	37	1.55B	2	0
30K	292,795	301	30K	2988	17	1.60B	2	0
40K	172,231	30	40K	1710	10	1.65B	2	0
50K	119,072	0	50K	1108	6	1.70B	2	0
60K	84,650	0	60K	793	5	1.75B	2	0
70K	66,239	0	70K	582	4	1.80B	1	0
80K	52,672	0	80K	439	3	1.85B	1	0
90K	42,799	0	90K	351	3	1.90B	1	0

Table 24
Number of discovered itemsets in dense databases at various μ thresholds.

Connect			**Accidents**			**Chess**		
μ	FEACP &CLH-Miner	HUIs	μ	FEACP &CLH-Miner	HUIs	μ	FEACP &CLH-Miner	HUIs
50.0M	12	0	187.0M	6	0	1.70M	811	0
50.1M	12	0	187.5M	6	0	1.75M	453	0
50.2M	6	0	188.0M	6	0	1.80M	243	0
50.3M	4	0	188.5M	6	0	1.85M	132	0
50.4M	4	0	189.0M	5	0	1.90M	65	0
50.5M	4	0	189.5M	5	0	1.95M	39	0
50.6M	4	0	190.0M	4	0	2.00M	18	0
50.7M	4	0	190.5M	4	0	2.05M	8	0
50.8M	4	0	191.0M	4	0	2.10M	5	0

from several levels of abstractions than traditional HUIs. In the case of Chainstore and Connect, the number of HUIs returned is zero across all the μ thresholds tested. It can be concluded that the results returned by traditional HUI algorithms may miss several meaningful itemsets which can be discovered by FEACP and CLH-Miner. For example, for $\mu = 750$K and the Fruithut dataset, FEACP and CLH-Miner were able to discover five CLHUIs, some of which contain items from different abstraction levels, such as $\{Cabbages, AsianVegies\}$, $\{Cabbages, BunchVegies\}$ and $\{AsianVegies, Cucumbers\}$, while the number of HUIs returned is zero, which completely missed the itemsets found by FEACP and CLH-Miner.

To evaluate the impact of the μ threshold on memory consumption, the peak memory usage of each algorithm was recorded. Fig. 4 presents the results. Generally, on all the tested databases the memory usage of FEACP is always lower than that of FEACP(lu), since the sub-tree utility is able to tighten the search space further than the local utility upper bound, and thus the low memory consumption. As in the runtime evaluations on the first two databases, Foodmart (Fig. 4a) and Fruithut (Fig. 4b), the memory usage of both FEACP and FEACP(lu) is much lower than that of CLH-Miner, thanks to the effective use of upper bounds to prune the search space. The memory usage of FEACP on Foodmart (Fig. 4a) is cut by 49% at $\mu = 10$Kwhen compared with CLH-Miner, and by 4% when compared with FEACP(lu). The same memory usage reduction is also observed on the Fruithut database (Fig. 4b). At $\mu = 10$K, FEACP reduced memory consumption by 2% when compared with FEACP(lu) and 55% when compared with CLH-Miner.

On the large-sparse Chainstore database, CLH-Miner has better memory consumption when compared against both FEACP and FEACP(lu). On average CLH-Miner consumed 6% less memory than FEACP and 11% less than FEACP(lu) across all μ thresholds. This can be explained by the fact that the number of itemsets that need to be stored in the utility-list of FEACP and FEACP(lu) is much higher than with CLH-Miner. This thus leads to 6% and 11% higher memory usage on average on FEACP and FEACP(lu), respectively, when compared to CLH-Miner.

With regard to the Connect database (Fig. 4d), which is a dense database, FEACP kept the memory usage as low as 12% of that of CLH-Miner for high μ thresholds, and up to 47% lower at $\mu = 50.3$M. After that, since the runtime of CLH-Miner exceeded three hours, no further tests were needed. FEACP(lu), which employs only local utility to prune candidates, obviously consumed more memory than FEACP in this test, at an average of 15% more across all thresholds. Considering the Accidents database in Fig. 4e, FEACP also has the lowest memory usage, which is 5% lower than CLH-Miner and 11% lower than FEACP(lu) on average. FEACP(lu) has shown that with only local utility applied its pruning efficiency is not as good compared to both FEACP and CLH-Miner in this database, hence the highest memory usage here. Regarding Chess (Fig. 4f), it is the smallest and the last of the tested databases. CLH-Miner managed to keep the memory usage lower than both FEACP and FEACP(lu) at high μ thresholds, by up to 8% compared to FEACP and 13% compared to FEACP(lu). This can be explained by the fact that CLH-Miner is capable of pruning more candidates than FEACP and FEACP(lu) at these thresholds. However, as the threshold was reduced further FEACP again showed its efficiency in candidate pruning, keeping the memory usage from 1% to 4% lower than CLH-Miner, starting from $\mu = 1.85$M. As seen in the runtime evaluation, FEACP has proven its effec-

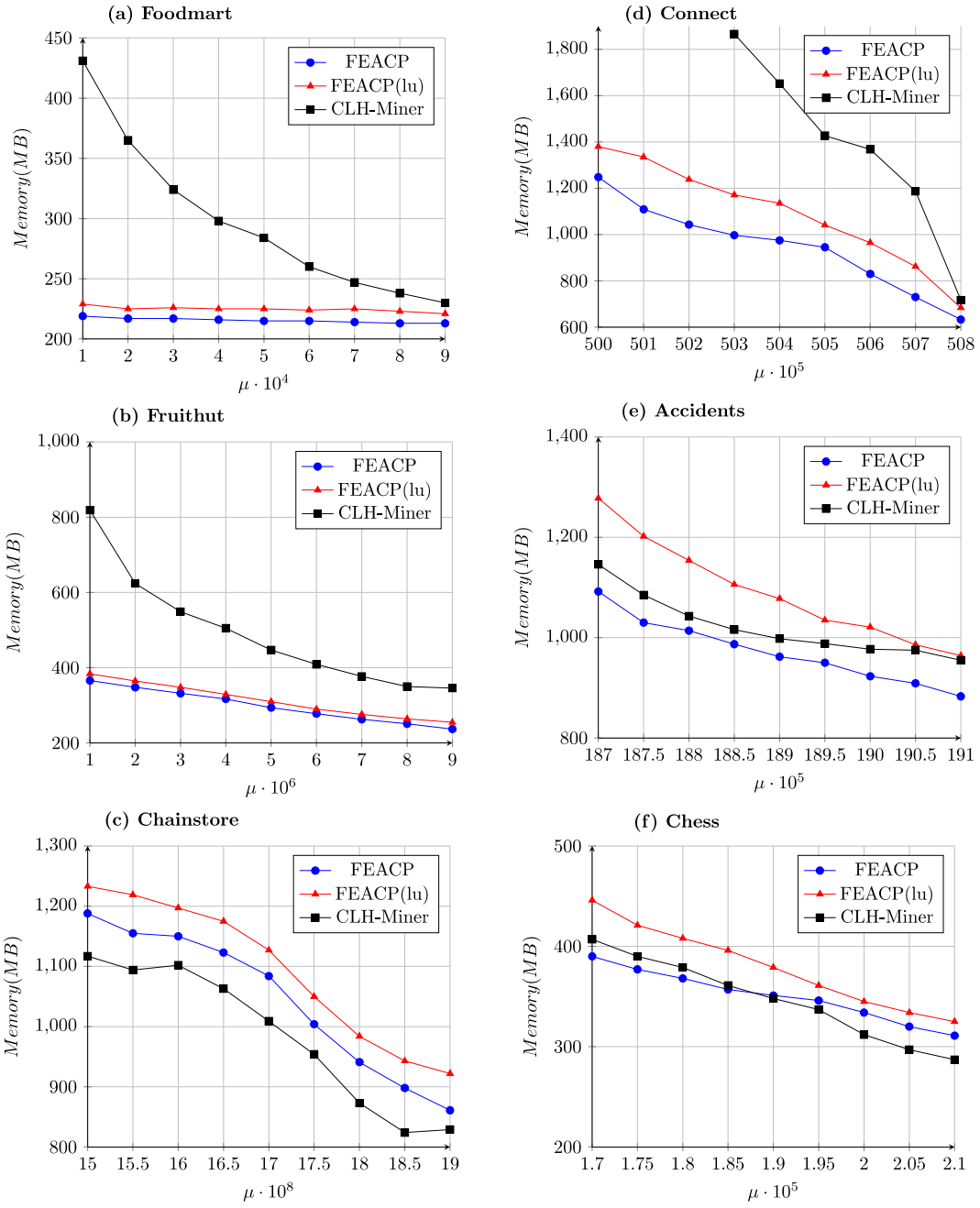

Fig. 4. Memory usage comparison on the test databases.

tiveness with dense databases. Overall, the proposed FEACP algorithm, which applies efficient pruning strategies, managed to keep the average memory usage low by eliminating unpromising candidates from the search space when mining cross-level itemsets.

Another reason why FEACP achieved a substantial speed boost and has more effective memory consumption is because it extends the framework of the EFIM algorithm [16], which is an efficient algorithm for HUIM. Using the highly effective upper bounds method to prune candidates from EFIM, FEACP can discard many more unpromising itemsets when compared to CLH-Miner and thus dramatically reduce the time and memory needed to mine CLHUIs. However, as pointed out in [18] the drawback of EFIM on large sparse databases is also the drawback of FEACP on such databases when compared to CLH-Miner, which is extended from the FHM algorithm [28].

6. Conclusion and future work

In this work, a novel algorithm named FEACP was presented to efficiently discover cross-level high-utility itemsets in quantitative databases. FEACP takes as input a quantitative transaction database with an item taxonomy, which describes categories and sub-categories of items. FEACP relies on upper bounds on the utility and pruning strategies. This makes it possible to reduce the number of itemsets that FEACP visits in the search space. FEACP adopts these techniques to discover the complete set of cross-level high-utility itemsets in quantitative transaction databases. Several experiments were conducted on both synthetic and real databases, which confirmed that FEACP can find insightful itemsets. Furthermore, through these empirical evaluations FEACP has been shown to have excellent performance in terms of mining time and memory consumption with respect to the previous state-of-the-art algorithm for the same problem, namely CLH-Miner. The execution time of FEACP was found to be up to 177 times less than that of CLH-Miner while returning the same set of discovered itemsets. Furthermore, the memory consumption of FEACP is about half that of CLH-Miner on the three test databases (Foodmart, Fruithut and Connect), thanks to the effective search space pruning strategies. However, the memory consumption of FEACP on large, sparse databases is higher than that of CLH-Miner in the case of the Chainstore database. For future works, we will focus on extending FEACP to mine closed or maximal CLHUI from transaction databases, applying more effective pruning strategies to address the drawback on large sparse databases. Parallel computing frameworks will be studied to reduce mining time, as well as be able to compute with larger databases.

CRediT authorship contribution statement

Thanh-Tung Nguyen: Methodology, Software, Visualization, Writing – original draft. **Loan T.T. Nguyen:** Writing – review & editing, Validation, Supervision. **Trinh D.D. Nguyen:** Software, Writing – original draft, Visualization. **Philippe Fournier-Viger:** Writing – review & editing, Resources. **Ngoc-Thanh Nguyen:** Validation, Writing – review & editing. **Bay Vo:** Writing – review & editing, Validation, Formal analysis.

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

References

[1] R. Agrawal, T. Imieliński, A. Swami, Mining Association Rules Between Sets of Items in Large Databases, ACM SIGMOD Record 22 (2) (1993) 207–216.
[2] P. Fournier-Viger, J. C. W. Lin, B. Vo, T. T. Chi, J. Zhang, and H. B. Le, "A survey of itemset mining," *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, vol. 7, no. 4, 2017.
[3] R. Agrawal and R. Srikant, "Fast Algorithms for Mining Association Rules in Large Databases," in *the 20th International Conference on Very Large Data Bases (VLDB '94)*, 1994, pp. 487–499.
[4] J. Han, J. Pei, Y. Yin, R. Mao, Mining frequent patterns without candidate generation: A frequent-pattern tree approach, Data Min. Knowl. Disc. 8 (1) (2004) 53–87.
[5] M.J. Zaki, K. Gouda, Fast vertical mining using diffsets, in: in *the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2003, pp. 326–335.
[6] T. Uno, T. Asai, Y. Uchida, H. Arimura, An Efficient Algorithm for Enumerating Closed Patterns in Transaction Databases, International Conference on Discovery Science 3245 (2004) 16–31.
[7] H. Yao, H.J. Hamilton, G.J. Butz, A foundational approach to mining itemset utilities from databases, SIAM International Conference on Data Mining 4 (2004) 482–486.
[8] H. Yao, H.J. Hamilton, Mining itemset utilities from transaction databases, Data Knowl. Eng. 59 (3) (2006) 603–626.
[9] Y. Liu, W. K. Liao, and A. Choudhary, "A two-phase algorithm for fast discovery of high utility itemsets," in *the 9th Pacific-Asia Conference on Advances in Knowledge Discovery and Data Mining*, 2005, vol. 3518, pp. 689–695.
[10] C.F. Ahmed, S.K. Tanbeer, B.S. Jeong, Y.K. Lee, Efficient tree structures for high utility pattern mining in incremental databases, IEEE Trans. Knowl. Data Eng. 21 (12) (Dec. 2009) 1708–1721.
[11] V.S. Tseng, C.W. Wu, B.E. Shie, P.S. Yu, UP-Growth: An efficient algorithm for high utility itemset mining, in: in *the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2010, pp. 253–262.
[12] M. Liu, J. Qu, Mining high utility itemsets without candidate generation, in: in *ACM International Conference Proceeding Series*, 2012, pp. 55–64.
[13] J. Liu, K. Wang, B.C.M. Fung, Direct discovery of high utility itemsets without candidate generation, in: in *Proceedings - IEEE International Conference on Data Mining*, 2012, pp. 984–989.
[14] J. Liu, K. Wang, B.C.M. Fung, Mining High Utility Patterns in One Phase without Generating Candidates, IEEE Trans. Knowl. Data Eng. 28 (5) (May 2016) 1245–1257.
[15] P. Fournier-Viger, C. W. Wu, S. Zida, and V. S. Tseng, "FHM: Faster high-utility itemset mining using estimated utility co-occurrence pruning," in *International Symposium on Methodologies for Intelligent Systems*, 2014, vol. 8502 LNAI, pp. 83–92.
[16] S. Zida, P. Fournier-Viger, J.C.W. Lin, C.W. Wu, V.S. Tseng, EFIM: a fast and memory efficient algorithm for high-utility itemset mining, Knowl. Inf. Syst. 51 (2) (2017) 595–625.
[17] S. Krishnamoorthy, HMiner: Efficiently mining high utility itemsets, Expert Syst. Appl. 90 (2017) 168–183.
[18] L.T.T. Nguyen, P. Nguyen, T.D.D. Nguyen, B. Vo, P. Fournier-Viger, V.S. Tseng, Mining high-utility itemsets in dynamic profit databases, Knowl.-Based Syst. 175 (2019) 130–144.
[19] J. Han, Y. Fu, Mining multiple-level association rules in large databases, IEEE Trans. Knowl. Data Eng. 11 (5) (1999) 798–805.
[20] B. Vo, B. Le, Fast Algorithm for Mining Generalized Association Rules, Int J. of Database Theory 2 (3) (Feb. 2009) 19–21.
[21] C.M. Wu, Y.F. Huang, Generalized association rule mining using an efficient data structure, Expert Syst. Appl. 38 (6) (Jun. 2011) 7277–7290.
[22] P. Rawat, S. Kant, B. Pant, A. Chaudhary, and S. K. Sharma, "A better approach for multilevel association rule mining," in *5th International Conference on Soft Computing for Problem Solving*, 2016, vol. 437, pp. 597–604.

[23] Y. Cheng, W. Der Yu, Q. Li, GA-based multi-level association rule mining approach for defect analysis in the construction industry, Autom. Constr. vol. 51, no. C (2015) 78–91.
[24] E. Baralis, L. Cagliero, T. Cerquitelli, P. Garza, Generalized association rule mining with constraints, Inf. Sci. 194 (2012) 68–84.
[25] R. Srikant, R. Agrawal, Mining generalized association rules, Future Generation Computer Systems 13 (2–3) (1997) 161–180.
[26] R.M. Juvenil Ayres, M.T. Prado Santos, FOntGAR algorithm: Mining generalized association rules using fuzzy ontologies, in: in *IEEE International Conference on Fuzzy Systems*, 2012, pp. 667–681.
[27] L. Cagliero, S. Chiusano, P. Garza, G. Ricupero, Discovering high-utility itemsets at multiple abstraction levels, European Conference on Advances in Databases and Information Systems 767 (2017) 224–234.
[28] P. Fournier-Viger et al, Mining Cross-Level High Utility Itemsets, in: in *33rd International Conference on Industrial, Engineering and Other Applications of Applied Intelligent Systems*, 2020, p. 12.
[29] P. Fournier-Viger, J. Chun-Wei Lin, T. Truong-Chi, R. Nkambou, A Survey of High Utility Itemset Mining, in: P. Fournier-Viger, J.-C.-.-W. Lin, R. Nkambou, B. Vo, V.S. Tseng (Eds.), High-Utility Pattern Mining: Theory, Algorithms and Applications, Springer International Publishing, Cham, 2019, pp. 1–45.
[30] W. Gan, J.C.W. Lin, P. Fournier-Viger, H.C. Chao, P.S. Yu, A survey of parallel sequential pattern mining, ACM Trans. Knowl. Discovery Data 13 (3) (Jun. 2019) 34.
[31] C. Zhang, G. Almpanidis, W. Wang, C. Liu, An empirical evaluation of high utility itemset mining algorithms, Expert Syst. Appl. 101 (2018) 91–115.
[32] S. Krishnamoorthy, Pruning strategies for mining high utility itemsets, Expert Syst. Appl. 42 (5) (2015) 2371–2381.
[33] L.T.T. Nguyen, D.B. Vu, T.D.D. Nguyen, B. Vo, Mining Maximal High Utility Itemsets on Dynamic Profit Databases, Cybernetics and Systems 51 (2) (2020) 140–160.
[34] P. Fournier-Viger, Y. Zhang, J. Chun-Wei Lin, H. Fujita, Y.S. Koh, Mining local and peak high utility itemsets, Inf. Sci. 481 (2019) 344–367.
[35] B. Vo et al, Mining Correlated High Utility Itemsets in One Phase, IEEE Access 8 (May 2020) 90465–90477.
[36] P. Fournier-Viger, Y. Yang, J.-C.-W. Lin, J. Frnda, Mining Locally Trending High Utility Itemsets, in: in *Pacific-Asia Conference on Knowledge Discovery and Data Mining*, 2020, pp. 99–111.
[37] J. Sahoo, A.K. Das, A. Goswami, An efficient approach for mining association rules from high utility itemsets, Expert Syst. Appl. 42 (13) (2015) 5754–5778.
[38] T. Mai, B. Vo, L.T.T. Nguyen, A lattice-based approach for mining high utility association rules, Inf. Sci. 399 (2017) 81–97.
[39] T. Mai, L.T.T. Nguyen, B. Vo, U. Yun, T.P. Hong, Efficient algorithm for mining non-redundant high-utility association rules, Sensors 20 (4) (2020) 1078.
[40] C.W. Wu, B.E. Shie, V.S. Tseng, P.S. Yu, Mining top-K high utility itemsets, in: in *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2012, pp. 78–86.
[41] S. Lee, J.S. Park, "Top-k high utility itemset mining based on utility-list structures", in, International Conference on Big Data and Smart Computing (BigComp) 2016 (2016) 101–108.
[42] J. Liu, X. Zhang, B.C.M. Fung, J. Li, F. Iqbal, Opportunistic mining of top-n high utility patterns, Inf. Sci. 441 (2018) 171–186.
[43] K. Singh, S.S. Singh, A. Kumar, B. Biswas, TKEH: an efficient algorithm for mining top-k high utility itemsets, Applied Intelligence 49 (3) (2019) 1078–1097.
[44] S. Krishnamoorthy, Mining top-k high utility itemsets with effective threshold raising strategies, Expert Syst. Appl. 117 (2019) 148–165.
[45] Y. Ou, Z.J. Liu, H.R. Karimi, Y. Tian, Multilevel association rule mining for bridge resource management based on immune genetic algorithm, Abstract and Applied Analysis 2014 (2) (2014) 1–8.
[46] M. Zhong, T. Jiang, Y. Hong, X. Yang, Performance of multi-level association rule mining for the relationship between causal factor patterns and flash flood magnitudes in a humid area, Geomatics, Natural Hazards and Risk 10 (1) (2019) 1967–1987.
[47] S. Ayubi, M.K. Muyeba, A. Baraani, J. Keane, An algorithm to mine general association rules from tabular data, Inf. Sci. 179 (20) (2009) 3520–3539.
[48] P. Manda, S. Ozkan, H. Wang, F. McCarthy, S.M. Bridges, Cross-Ontology Multi-level Association Rule Mining in the Gene Ontology, PLoS ONE 7 (10) (2012) 1–9.
[49] J. Hipp, A. Myka, R. Wirth, U. Güntzer, A new algorithm for faster mining of generalized association rules, European Symposium on Principles of Data Mining and Knowledge Discovery 1510 (1998) 74–82.
[50] K. Sriphaew, T. Theeramunkong, A new method for finding generalized frequent itemsets in generalized association rule mining, in: in *IEEE Symposium on Computers and Communications*, 2002, pp. 1040–1045.
[51] M.J. Zaki, C.J. Hsiao, Efficient algorithms for mining closed itemsets and their lattice structure, IEEE Trans. Knowl. Data Eng. 17 (4) (2005) 462–478.
[52] I. Pramudiono, M. Kitsuregawa, FP-tax: Tree structure based generalized association rule mining, in: in *Proceedings of the ACM SIGMOD International Conference on Management of Data*, 2004, pp. 60–63.
[53] C.L. Lui, F.L. Chung, Discovery of generalized association rules with multiple minimum supports, European Conference on Principles of Data Mining and Knowledge Discovery 1910 (2000) 510–515.
[54] M. Nouioua, Y. Wang, P. Fournier-Viger, J.-C.-W. Lin, J.-M.-T. Wu, "TKC: Mining Top-K Cross-Level High Utility Itemsets", in, International Conference on Data Mining Workshops (ICDMW) 2020 (2020) 673–682.