Mining Cross-Level High Utility Itemsets

Philippe Fournier-Viger[1], Ying Wang[2],
Jerry Chun-Wei Lin[3], Jose Maria Luna[4], and Sebastian Ventura[4]

[1] School of Natural Sciences and Humanities,
Harbin Institute of Technology (Shenzhen), Shenzhen, China
[2] School of Computer Sciences and Technology,
Harbin Institute of Technology (Shenzhen), Shenzhen, China
[3] Department of Computing, Mathematics and Physics, Western Norway University
of Applied Sciences (HVL), Bergen, Norway
[4] Department of Computer Sciences, University of Cordoba, Cordoba, Spain
philfv@hit.edu.cn, iwangying_919@163.com,
jerrylin@ieee.org, jmluna@uco.es, sventura@uco.es

Abstract. Many algorithms have been proposed to find high utility itemsets (sets of items that yield a high profit) in customer transactions. Though, it is useful to analyze customer behavior, it ignores information about item categories. To consider a product taxonomy and find high utility itemsets describing relationships between items and categories, the ML-HUI Miner was recently proposed. But it cannot find cross-level itemsets (itemsets mixing items from different taxonomy levels), and it is inefficient as it does not use relationships between categories to reduce the search space. This paper addresses these issues by proposing a novel problem called cross-level high utility itemset mining, and an algorithm named CLH-Miner. It relies on novel upper bounds to efficiently search for high utility itemsets when considering a taxonomy. An experimental evaluation with real retail data shows that the algorithm is efficient and can discover insightful patterns describing customer purchases.

Keywords: High Utility Mining · Cross-Level Itemset · Taxonomy.

1 Introduction

Pattern mining is a sub-field of data mining that aims at discovering interesting patterns in data to better understand the data or support decision-making. One of the most popular pattern mining tasks is frequent itemset mining (FIM), which consists of finding all frequently co-occurring itemsets (sets of values) in a customer transaction database [**?**,14]. Although FIM is useful, it does not consider purchase quantities of items and their relative importance. To address this issue, a more general problem was proposed called high utility itemset mining (HUIM) [5, 6, 12, 18], where items in transactions have purchase quantities and items have weights indicating their relative importance. The goal of HUIM is to find itemset that have a high utility (importance) such as those that yield a high profit in a customer transaction database. As HUIM is more general than

FIM, it is also more difficult. The reason is that the utility measure is not anti-monotonic (the utility of an itemset may be larger, smaller thann or equal to that of its subsets). Thus, FIM techniques cannot be directly used in HUIM.

Although HUIM has many applications, it ignores that items are often organized in a taxonomy. For example, two items *dark chocolate* and *milk chocolate* are both specializations of the abstract item *chocolate*, which can in turn be a specialization of an item *snack and treats*. Traditional HUIM can only find patterns involving items at the lowest abstraction level. This is a major problem because although some items like *milk chocolate* and *dark chocolate* may not appear in HUIs, the item *chocolate* could be part of some HUIs. Thus, important information may be missed by traditional HUIM algorithms.

To mine frequent itemsets that contain items of different abstraction levels, many algorithms were developed [1, 9, 11, 13, 16, 20, 21]. But these algorithms all rely on the fact that the support (frequency) measure is anti-monotonic to reduce the search space. But this property does not hold for the utility. Hence, it is an important but difficult challenge to mine high utility itemsets containing items of different abstraction levels. Recently, a first work has been done in this direction by Cagliero et al. [2]. They proposed an algorithm named ML-HUI Miner to mine HUIs where items are of different abstraction levels. However, the algorithm has two important limitations. First, it can only find HUIs where all items are of the same abstraction level. This assumption makes the problem easier to solve but results in missing all patterns where items are of different levels, which are often interesting. Second, the algorithm utilizes simple properties to reduce the search space. It mines the different abstraction levels independently and does not use the relationships between abstraction levels to reduce the search space.

To address these limitations, this paper defines the more general problem of mining all cross-level high utility itemsets and studies its properties, and an algorithm named CLH-Miner is proposed to find all these itemsets efficiently. The algorithm relies on novel upper bounds and pruning properties to reduce the search space using relationships between items of different abstraction levels. Experiments on transaction data collected from two chains of retail stores show that interesting patterns are discovered, and that the algorithm's optimizations considerably improve its performance.

The rest of this paper is organized as follows. Section 2 reviews related work. Section 3 defines the problem of cross-level HUIM. Section 4 presents CLH-Miner. Section 5 describes experiments. Lastly, Section 6 draws a conclusion.

2 Related work

Several algorithms were designed to discover frequent itemsets in transaction databases [**?**]. FIM was then generalized as HUIM to find itemsets of high importance (e.g. profit) in datasets where items are annotated with purchase quantities and weights (e.g. unit profit) [5]. Some representative HUIM algorithms are Two-Phase [12], HUI-Miner/HUI-Miner* [18] and FHM [6]. Two-Phase [12] adopts a breadth-first search to explore the search space of itemsets and an up-

per bound on the utility measure called TWU that is anti-monotonic to reduce the search space. But Two-Phase has two important drawbacks: (1) the TWU upper bound is loose, and (2) Two-Phase can perform numerous database scans. HUI-Miner [18] addressed these limitations by introducing the tighter remaining utility upper bound and relying on a novel vertical structure called utility-list. HUI-Miner avoids performing numerous database scans by directly joining utility-lists to calculate the utility and upper bound values of itemsets. An improved version of HUI-Miner called FHM was then proposed [6] and developing efficient HUIM algorithms is an active research area [5]. Although traditional HUI mining algorithms are useful to identify important (e.g. profitable) patterns, most of these algorithms ignore item taxonomies. Thus, they are unable to reveal insightful relationships between items of different taxonomy levels (e.g. product categories).

In FIM, several studies have aimed at extracting generalized patterns using a taxonomy. Srikant and Agrawal [20] first proposed using a taxonomy of items linked by *is-a* relationships to extract frequent itemsets and association rules containing items from different abstraction levels. The Cumulate algorithm [20] proposed in that study requires that the user sets a minimum support threshold. Then, Cumulate performs a breadth-first search starting from single items to generate larger itemsets. Cumulate can find *cross-level patterns*, that is patterns containing items from different abstraction levels, with the constraint that an item and its taxonomy ancestor may not appear together in a pattern. Then, Hipp et al. proposed the Prutax algorithm [11] to more efficiently find cross-level frequent itemsets using a depth-first search and a vertical database format. Prutax uses two search space pruning strategies: eliminating an itemset from the search space if one of its subsets is infrequent or if it has a taxonomy ancestor that is infrequent. Another depth-first search algorithm called SET was proposed [21] for cross-level FIM, which was claimed to outperform Prutax. Han and Fu [9] proposed a variation of the above problem called *multi-level pattern mining*, where a different minimum support threshold can be set for each taxonomy level but items in a frequent itemset must be from the same taxonomy level. They designed breadth-first search algorithms that start from abstract itemsets and recursively specialize frequent itemsets. Lui and Chung [13] then proposed a variation to mine cross-level itemsets using multiple thresholds. Their algorithm performs a breadth-first search and finds generalized itemsets by recursively generalizing infrequent itemsets based on a concept of item taxonomy distance. Ong et al. [16] then proposed a FP-tree based pattern-growth algorithm for multi-level itemset mining using a concept of taxonomy-based quantities. Another FP-tree based algorithm was proposed by Pramudiono for cross-level itemset mining [17], and an algorithm based on the AFOPT-tree structure [15] was designed. Rajkumar et al. [19] proposed using different thresholds for different taxonomy levels and itemset lengths in multi-level itemset mining. Other variations of the above problems have also been studied such as to find misleading multi-level itemsets [1], concise representations of generalized itemsets [10], and mining multi-level association rules using evolutionary algorithms [23].

To our knowledge only one algorithm named ML-HUI Miner was proposed to consider taxonomies in high utility itemset mining [2]. This algorithm extends HUI-Miner but has several important drawbacks: (1) it is unable to find cross-level patterns (containing items from multiple taxonomy levels), (2) it mines each taxonomy levels independently, and (3) it does not use relationships between taxonomy levels for search space pruning with the utility measure. This paper addresses these issues by defining a more general problem of cross-level HUIM and an efficient algorithm named CLH-Miner to find the desired patterns.

3 Preliminaries and Problem Definition

This section introduces key definitions and then defines the proposed problem.

Definition 1 (Transaction database). *Let $I = \{i_1, i_2, \ldots, i_m\}$ be a set of items. A* transaction database *is a multiset of transactions $D = \{T_1, T_2, \ldots, T_n\}$ such that for each transaction T_c, $T_c \in I$, and T_c has a unique identifier c called its Tid. Each item $i \in I$ is associated with a positive number $p(i)$, called its external utility (e.g. unit profit). For each item $i \in T_c$, a positive number $q(i, T_c)$ is called the internal utility of i (e.g. purchase quantity of i in T_c).*

Definition 2 (Taxonomy). *A **taxonomy** τ is a directed acyclic graph (a tree) defined for a database D. It contains a leaf for each item $i \in I$. An inner node represents 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 i, j in τ represents an is-a relationship. Inner nodes are called **generalized items** or **abstract items**. The set of all generalized items is denoted as GI, and the set of all generalized or non-generalized items is denoted as $AI = GI \cup I$. Also, let there be a relation $LR \subseteq GI \times I$ such that $(g, i) \in LR$ iff there is a path from g to i. And, let there be a relation $GR \subseteq AI \times AI$ such that $(d, f) \in GR$ iff there is a path from d to f. An **itemset** X is a set of items such that $X \subseteq AI$ and $\nexists i, j \in X | i \in Desc(j, \tau)$. An itemset X is a **generalized itemset** iff $\exists g \in X$ such that $g \in GI$.*

Example 1. Table 1 depicts a database, which will be used as running example, having seven transactions $(T_1, T_2, ..., T_7)$ and five items $(I = \{a, b, c, d, e\})$, where internal utilities (e.g. quantities) are shown as integers beside items. For instance, transaction T_3 indicates that 1, 5, 1, 3 and 1 units of items a, b, c, d, e were bought, respectively. Table 2 indicates that the external utilities (unit profits) of these items are 5, 2, 1, 2, 3. Fig 1 depicts an item taxonomy for this database. For instance, items a and b are aggregated into the generalized item Y.

Definition 3 (Descendant/Specialization/Sibling). *The **leaf items** of a generalized item g in a taxonomy τ are all the leaves that can be reached by following paths starting from g. This set is formally defined as $Leaf(g, \tau) = \{i | (g, i) \in LR\}$. The **descendant items** of a (generalized) item d is the set $Desc(d, \tau) = \{f | (d, f) \in GR\}$. An itemset X is called a **descendant itemset***

Table 1. A transaction database

TID	Transaction
T_1	(a,1),(c,1)
T_2	(e,1)
T_3	(a,1),(b,5),(c,1),(d,3),(e,1)
T_4	(b,4),(c,3),(d,3),(e,1)
T_5	(a,1),(c,1),(d,1)
T_6	(a,2),(c,6),(e,2)
T_7	(b,2),(c,2),(e,1)

Table 2. External utility values

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

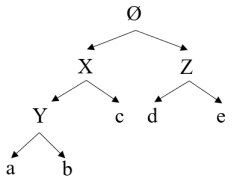

Fig. 1. A taxonomy of items

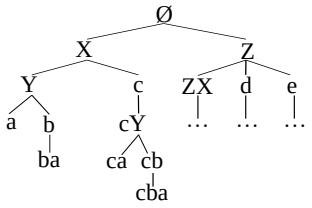

Fig. 2. A part of the search space

of an itemset Y if $|X| = |Y|$ and $\forall f \in X, \exists d \in Y | f \in Desc(d, \tau)$. An itemset X is called a ***specialization*** *of an itemset Y if $|X| = |Y|$ and $\forall f \in X, f \in Y \vee \exists d \in Y | f \in Desc(d, \tau)$. Two items a and b are said to be* ***siblings*** *if they have the same parent in τ. Furthermore, let $level(d)$ denotes the number of edges to be traversed to reach an item d starting from the root of τ.*

Example 2. In the taxonomy of Fig. 1, $Leaf(\{X\}, \tau) = \{a, b, c\}$ and $Desc(\{X\}, \tau) = \{Y, a, b, c\}$, The itemset $\{Y, d\}$ is a descendant of $\{X, Z\}$, and a specialization of $\{Y, Z\}$. Moreover, the items Y and c are siblings, and $level(Y) = 2$.

Definition 4 (Utility of an item/itemset). *The utility of an item i in a transaction T_c is denoted as $u(i, T_c)$ and defined as $p(i) \times q(i, T_c)$. The utility of an itemset X (a group of items $X \subseteq I$) in a transaction T_c is denoted as $u(X, T_c)$ and defined as $u(X, T_c) = \sum_{i \in X} u(i, T_c)$. The utility of an itemset X (in a database) is denoted as $u(X)$ and defined as $u(X) = \sum_{T_c \in g(X)} u(X, T_c)$, where $g(X)$ is the set of transactions containing X [12].*

Example 3. The utility of a in T_3 is $u(a, T_3) = 5 \times 1 = 5$. The utility of $\{a, c\}$ in T_3 is $u(\{a, c\}, T_3) = u(a, T_3) + u(c, T_3) = 5 \times 1 + 1 \times 1 = 6$. The utility of $\{a, c\}$ in the database is $u(\{a, c\}) = u(a) + u(c) = u(a, T_1) + u(a, T_3) + u(a, T_5) + u(a, T_6) + u(c, T_1) + u(c, T_3) + u(c, T_5) + u(c, T_6) = 5 + 5 + 5 + 10 + 1 + 1 + 1 + 6 = 34$.

Definition 5 (Utility of a generalized item/itemset). *The utility of a generalized item g in a transaction T_c is denoted and defined as $u(g, T_c) = \sum_{i \in Leaf(g, \tau)} p(i) \times q(i, T_c)$. The utility of a generalized itemset GX in a transaction T_c is defined as $u(GX, T_c) = \sum_{d \in GX} u(d, T_c)$. The utility of a generalized itemset GX (in a database) is denoted and defined as $u(GX) = \sum_{T_c \in g(GX)} u(GX, T_c)$, where $g(GX) = \{T_c \in D | \exists X \subseteq T_c \wedge X$ is a descendant of $GX\}$.*

Example 4. The utility of the generalized item Y in T_3 is $u(Y, T_3) = u(a, T_3) + u(b, T_3) = 1 \times 5 + 5 \times 2 = 15$. The utility of the generalized itemset $\{Y, c\}$ in T_3 is $u(\{Y, c\}, T_3) = u(Y, T_3) + u(c, T_3) = 15 + 1 = 16$. The utility of the generalized itemset $\{Y, c\}$ is $u(\{Y, c\}) = u(\{Y, c\}, T_1) + u(\{Y, c\}, T_3) + u(\{Y, c\}, T_4) + u(\{Y, c\}, T_5) + u(\{Y, c\}, T_6) + u(\{Y, c\}, T_7) = 6 + 16 + 11 + 6 + 16 + 6 = 61$.

Definition 6 (Cross-level high utility mining). *The problem of* ***cross-level high utility mining*** *is defined as finding all cross-level high-utility itemsets (CLHUIs). A (generalized) itemset X is a CLHUI if $u(X) \geq minutil$ for a user-specified minutil threshold.*

Example 5. If $minutil = 60$, the cross-level high utility itemsets in the database of the running example are $\{X\}, \{Y, c\}, \{Z, X\}, \{Z, Y\}, \{Z, Y, c\}, \{e, X\}, \{e, Y, c\}$ with respectively a utility of $61, 61, 84, 71, 84, 64, 64$.

4 Proposed Algorithm

To efficiently discover all cross-level HUIs, a novel algorithm is proposed, named CLH-Miner. It assumes that a processing order $\succ$ is defined on items of AI. That total order $\succ$ is defined such that $a \prec b$ for two items $a, b \in AI$, if $level(a) < level(b)$ or $level(a) = level(b) \wedge GWU(a) < GWU(b)$, where GWU is a utility measure. This ordering between levels ensures that the algorithm considers items that are higher in the taxonomy before those that are lower, which is important as it enables to prune specializations of itemsets using techniques that will be presented in this section. The algorithm explores the search space of all itemsets by recursively performing two types of extensions on itemsets, defined as follows.

Definition 7 (Extension). *Let X be an itemset. The* ***join-based extensions*** *of X are the itemsets obtained by appending an item y to X such that $y \in AI$, $y \succ i$, $\forall i \in X, y \notin Desc(i, \tau)$. The* ***tax-based extensions*** *of X are the itemsets obtained by replacing the last item y of X according to $\succ$ by a descendant of y.*

For example, if $Z \succ X$, the itemset $\{X, d\}$ is a tax-based extension of $\{X, Z\}$, which is a join-extension of $\{X\}$ with item Z. A part of the search space of the running example is shown in Fig. 2.

To quickly calculate the utility of an itemset and those of its extensions, the CLH-Miner algorithm creates a *tax-utility-list* structure for each itemset visited in the search space. This vertical structure extends the utility-list structure used in traditional HUIM [18] with taxonomy information, and is defined as follows.

Definition 8 (Tax-Utility-list). *Let there be an itemset X, a database D, and the total order $\succ$. The* ***tax-utility-list*** *$tuList(X)$ of X contains a tuple $(tid, iutil, rutil)$ for each transaction T_{tid} containing X. The iutil element of a tuple is the utility of X in T_{tid}. i.e., $u(X, T_{tid})$. The rutil element of a tuple is defined as $\sum_{i \in T_{tid} \wedge i \succ x \forall x \in X} u(i, T_{tid})$. Also, $tuList(X)$ contains childs, a set of pointers to the tax-utility-list of the child items of X in the taxonomy.*

For example, assume that $X \prec Z \prec Y \prec d \prec b \prec a \prec e \prec c$. The tax-utility-list of itemset $\{X, d\}$ is $tuList(\{X, d\}) = \{(3, 22, 3), (4, 17, 3), (5, 8, 0)\}$.

The proposed algorithm initially constructs the tax-utility-lists of single items by scanning the database, and then applies a modified version of the construct procedure of HUI-Miner [18] to create the tax-utility-lists of their extensions by joining the tax-utility-lists of other itemsets. The difference between the construct procedure of CLH-Miner and that of HUI-Miner is that *childs* is updated. A tax-utility list of an itemset X allows to quickly obtain the utility of X without scanning the database, as $u(X) = \sum_{e \in tuList(X)} e.iutil$ [18]. For example, based on $tuList(\{X, d\})$, the utility of $\{X, d\}$ is $u(\{X, d\}) = 22 + 17 + 8 = 47$.

By doing an exhaustive search of all transitive extensions of single items and calculating their utility using their tax-utility-lists, all cross-level HUIs can be found. But this approach is time-consuming if the number of items is large, which is the case for many real-life databases. To efficiently find cross-level HUIs, two search space pruning techniques are introduced next. The first one is a novel technique generalizing the TWU measure used in traditional HUIM [12].

Definition 9 (The GWU measure). *The* ***transaction utility*** *(*TU*) of a transaction T_c is the sum of the utilities of all the items in T_c. i.e. $TU(T_c) = \sum_{x \in T_c} u(x, T_c)$. The* ***generalized-weighted utilization*** *(GWU) of a (generalized) itemset $X \subseteq AI$ is defined as the sum of the transaction utilities of transactions containing X, i.e. $GWU(X) = \sum_{T_c \in g(X)} TU(T_c)$.*

Property 1 (The GWU is a monotone upper bound on the utility measure). Let there be two itemsets X and Y such that Y is a specialization of a superset of X. The GWU of X is no less than its utility ($GWU(X) \geq u(X)$). Moreover, the GWU of X is no less than the utility of its supersets ($GWU(X) \geq u(Y) \forall Y \supset X$).

Proof. If $Y \subset X$, then $u(Y) \leq GWU(Y) \leq GWU(X) < minutil$. If Y is a specialization of X, then $u(Y) \leq u(X) \leq GWU(X) < minutil$.

To reduce the search space, the next property is derived from Property 1.

Property 2 (Pruning the search space using the GWU). For any itemset X, if $GWU(X) < minutil$, then X is a low utility itemset as well as all its join-based and tax-based extensions.

For example, if $minutil = 50$, $GWU(\{c, d, e\}) = 45 < minutil$, and thus $\{c, d, e\}$ and all its extensions can be ignored, as they are not CLHUIs.

The third pruning technique utilizes information from the tax-utility-list of an itemset to prune its transitive extensions. This technique generalizes the remaining utility pruning technique of HUI-Miner for a taxonomy.

Property 3 (Pruning the search space using the remaining utility). Given the total order $\prec$ on a taxonomy, the *remaining utility upper bound* of an itemset X is the sum of the *iutil* and *rutil* values in its tax-utility-list. Formally, this upper bound is defined as $reu(X) = \sum_{e \in tuList(X)} (e.iutil + e.rutil)$. If $reu(X) < minutil$, then X and all its tax-based extensions are not CLHUIs.

Example 6. If $minutil = 25$, $reu(\{c, d, e\}) = 19 < minutil$, and thus $\{c, d, e\}$ and all its tax-based extensions can be ignored, as they are not CLHUIs.

The proposed CLH-Miner algorithm takes as input a transaction database with utility, a taxonomy τ, and the user-defined *minutil* threshold. CLH-Miner outputs the set of all CLHUIs. The pseudocode is shown in Algorithm 1. The algorithm first reads the database to calculate the GWU of each item and generalized item. Then, the algorithm identifies the set I^* of each item i whose GWU is no less than *minutil* and for which $GWU(g) \geq minutil$ if g is a generalized item such that $i \in Desc(g, \tau)$. Next the algorithm identifies the set GI^* of all generalized items having a GWU no less than *minutil*. Thereafter, all items not in I^* and GI^* and their extensions are ignored according to pruning Property 2. Then, item taxonomy levels and GWU values are used to calculate the total order $\prec$ on items (items first sorted by ascending taxonomy levels, and then by ascending GWU values). A second database scan is then performed to reorder items in transactions according to the $\succ$ order, and build the tax-utility-lists of each item $i \in I^*$ and generalized item $g \in GI^*$. Then, the depth-first search of itemsets starts by calling the recursive *Search* procedure with the set $TULs$ of generalized items of taxonomy level 1, and the *minutil* threshold.

Algorithm 1: The CLH-Miner algorithm

input : D: a transaction database, τ: a taxonomy, *minutil*: the threshold
output: the CLHUIs

1 Scan D and τ to calculate the GWU of each item $i \in I$ and each generalized item $g \in GI$;
2 $I^* \leftarrow \{i | i \in I \wedge GWU(i) \geq minutil \wedge \forall g \in GI$ such that $Desc(g, \tau) \ni i$, $GWU(g) \geq minutil\}$;
3 $GI^* \leftarrow \{g | g \in GI \wedge GWU(g) \geq minutil\}$;
4 Calculate the total order $\prec$ on $I^* \cup GI^*$;
5 Scan D and τ to build the *tuList* of each item $i \in I^*$ and of generalized item $g \in GI^*$;
6 $TULs \leftarrow \{g | g \in GI^* \wedge level(g) = 1\}$;
7 `Search` ($TULs$, *minutil*);

The *Search* procedure takes as input (1) $PULs$, a set of itemsets of the form Px that extend an itemset P with some item x (initially $P = \emptyset$), and (2) the user-specified *minutil* threshold. The search procedure is applied as follows. For each itemset $Px \in TULs$, if the sum of the *iutil* values of the $tuList(Px)$ is no less than *minutil*, then Px is a CLHUI and it is output. Then, *join-based extensions* of Px are generated by joining Px with all extensions Py of $TULs$ such that $x \succ y$ to generate extensions of the form Pxy containing $|Px| + 1$ items. If the GWU of the *join-based extensions* is larger or equal than *minutil*, they are added to $ExtensionsOfX$. The $tuList(Pxy)$ is then constructed by applying a modified version of the *Construct* procedure of HUI-Miner to join the $tuList(P)$, $tuList(Px)$ and $tuList(Py)$. The difference with the original *Construct* procedure is that the *tax-utility-list* is added as a child item of y. This procedure is not shown due to the space limitation. Then, if the sum of *iutil* and *rutil* values in the $tuList(Px)$ is no less than *minutil*, *tax-based-extensions* of Px should be

Algorithm 2: The *Search* procedure

```
   input  : TULs: a set of extensions of an itemset P, minutil: the threshold
   output: the CLHUIs that are transitive extensions of P
 1 foreach itemset X ∈ TULs do
 2 |   if SUM(X.tuList.iutils) ≥ minutil then Output X;
 3 |   ExtensionsOfX ← ∅;
 4 |   foreach itemset Y ∈ TULs such that X ≺ Y do
 5 |   |   JoinExtension.tuList ← Construct(X, Y);
 6 |   |   if GWU(JoinExtension) ≥ minutil then
   |   |     ExtensionsOfX ← ExtensionsOfX ∪ {JoinExtension};
 7 |   end
 8 |   if SUM(X.tuList.iutils)+SUM(X.tuList.rutils) ≥ minutil then
 9 |   |   foreach itemset T ∈ X.childs do
10 |   |   |   TaxExtension.tuList ← Construct(X, T);
11 |   |   |   if GWU(TaxExtension) ≥ minutil then
   |   |   |     ExtensionsOfX ← ExtensionsOfX ∪ {TaxExtension};
12 |   |   end
13 |   end
14 |   Search (ExtensionsOfX,minutil);
15 end
```

explored (by Property 3). This is done by replacing item x with each child item of x. Also, if the *GWU* of *tax-based extensions* is larger or equal to *minutil*, they are added to $ExtensionsOfX$. Then, a recursive call to the *Search* procedure with extensions of Px is done to calculate its utility and explore its extension(s). When the algorithm terminates all the cross-level high utility itemsets have been output. Since the algorithm only eliminate itemsets using Properties 2 and 3, it can be proven that the algorithm outputs all CLHUIs.

5 Experiment

To evaluate CLH-Miner's performance, experiments were done on a computer equipped with an Intel(R) Xeon(R) CPU W-2123 3.60GHz, 32 GB of RAM, and running Windows 10. We compared the performance of CLH-Miner for CLHUI mining with ML-HUI Miner for multi-level HUIM, and HUI-Miner for HUIM. Algorithms were implemented in Java, and runtime and memory were measured using the standard Java API. Two real-world datasets were used, named *Liquor* ($|D|$ = 9,284, $|I|$ = 2,626, $|GI|$ = 77, T_{max} = 5, T_{avg} = 2.7, $Level$ = 7) and *Fruithut* ($|D|$ = 181,970, $|I|$ = 1,265, $|GI|$ = 43, T_{max} = 36, T_{avg} = 3.58, $Level$ = 4), where $|D|$ is the transaction count, $|I|$ is the item count, $|GI|$ is the generalized item count, T_{max} is the maximum transaction length, T_{avg} is the average transaction length, and $Level$ is the maximum level. *Liquor* and *Fruithut* contain transactions from US liquor stores and grocery stores, respectively.

In a first experiment, *minutil* was varied to evaluate its influence on the performance of CLH-Miner. Three versions of CLH-Miner were compared: (1) CLH-Miner, (2) CLH-Miner without the GWU pruning strategy, and (3) CLH-Miner without the remaining utility pruning strategy. Results for runtime and peak memory usage are shown in Fig. 3 for the two datasets. It is found that as *minutil* increases runtime decreases. This is reasonable since the lower *minutil*

is, the more CLHUIs are found, and the larger the search space is. For example, on *Liquor*, for *minutil* values of 40, 000 and 50, 000, there are 939 and 469 CLHUIs, and the runtimes are 47.76 and 19.83 seconds, respectively. It is also found that *GWU*-based pruning generally greatly decreases runtime, and memory usage. For instance, on *Fruithut*, when $minutil = 20,000,000$, CLH-Miner with *GWU*-based pruning is up to 3.74 times faster than CLH-Miner without it and uses up to 1.68 times less memory. Also, it is found that the *remaining utility* pruning strategy reduces memory usage, and slightly reduces runtime. For example, on *Fruithut*, when $minutil = 25,000,000$, memory is reduced by up to 47%. The *remaining utility* pruning strategy reduces runtime and memory because when the strategy is applicable, less extensions are stored in memory.

We also evaluated the proposed algorithm's scalability in terms of runtime and pattern count when transaction count is varied. For this experiment, *Fruithut* was divided into five parts and the algorithm's performance was measured after adding each part to the previous ones. Fig. 4 shows the results for $minutil = 10,000,000$. It is found that runtime and pattern count increase with database size. This is because the utility of itemsets may be greater in a larger database, which increases the number of tax-utility-list to be created for itemsets, and more time is needed to search a larger search space.

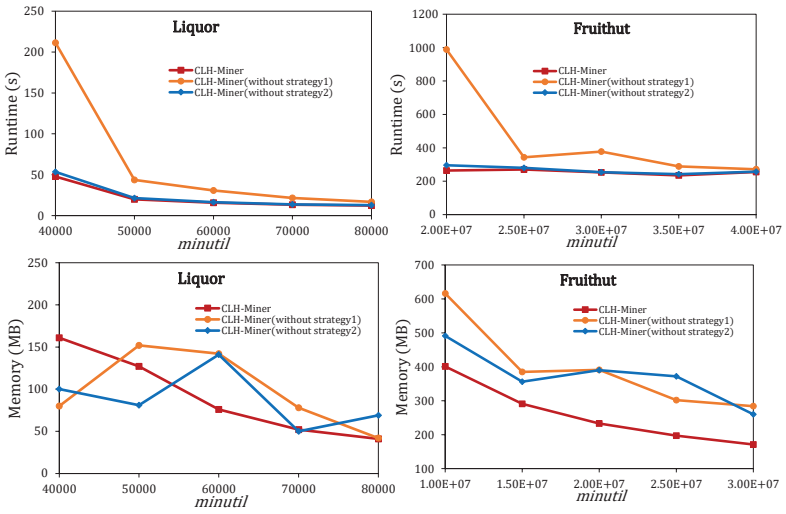

Fig. 3. Runtime and peak memory usage of three versions of CLH-Miner

In a third experiment, CLH-Miner's performance was compared with that of HUI-Miner and ML-HUI Miner. Table 3 and 4 show the runtime and peak memory usage for the two datasets. It is found that CLH-Miner takes more time than HUI-Miner and ML-HUI Miner, and that CLH-Miner generally consumes more memory than HUI-Miner and ML-HUI Miner on *Liquor*, and less memory

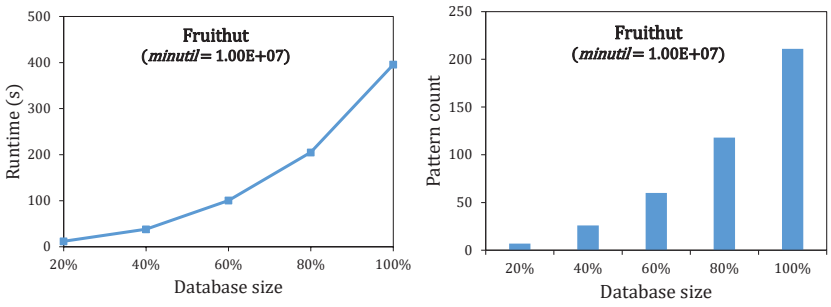

Fig. 4. Scalability of CLH-Miner when variying the database size

than ML-HUI Miner on *Fruithut*. This is reasonable since the proposed problem of CLHUI mining has a much larger search space than HUIM and multi-level HUIM. Thus, more patterns are considered and kept in memory.

Table 3. Comparison of CLH-Miner, ML-HUI Miner and HUI-Miner on *Liquor*

minutil	Runtime (s)			Memory (MB)		
	CLH-Miner	ML-HUI Miner	HUI-Miner	CLH-Miner	ML-HUI Miner	HUI-Miner
9k	877.74	39.23	0.08	222	130	59
10k	738.77	40.65	0.08	146	67	104
11k	515.81	39.00	0.06	183	111	27
12k	448.82	39.76	0.05	229	180	73
13k	406.77	40.74	0.05	150	316	118

Table 4. Comparison of CLH-Miner, ML-HUI Miner and HUI-Miner on *Fruithut*

minutil	Runtime (s)			Memory (MB)		
	CLH-Miner	ML-HUI Miner	HUI-Miner	CLH-Miner	ML-HUI Miner	HUI-Miner
4E+06	1,198.20	76.36	1.43	938	378	140
5E+06	1,166.74	83.26	1.16	778	1,788	129
6E+06	806.20	92.22	1.03	668	1,892	121
7E+06	694.89	97.96	0.69	581	2,162	100
8E+06	603.69	96.70	0.55	520	1,803	92

The number of pattern found by the algorithms was also compared for different *minutil* values. Results are shown in Table 5. It is observed that CLH-Miner finds more patterns than HUI-Miner (HUIs) and ML-HUI Miner (Multi-level HUIs). For example, for *Liquor* and $minutil = 50,000$, the number of CLHUIs is 9.57 times and 39.08 times greater than that of multi-level HUIs and HUIs, respectively. Hence, CLH-Miner finds some patterns that HUI-Miner and ML-HUI

Table 5. Comparison of pattern count on *Liquor* and *Fruithut*

Liquor				*Fruithut*			
minutil	CLHUIs	Multi-level HUIs	HUIs	*minutil*	CLHUIs	Multi-level HUIs	HUIs
10k	5537	234	241	4E+06	1248	55	11
20k	2136	119	86	5E+06	833	45	8
30k	1391	79	43	6E+06	586	36	4
40k	939	56	19	7E+06	435	30	2
50k	469	49	12	8E+06	331	23	1

Miner cannot find. For example, for $minutil = 500,000$, CLH-Miner can find interesting HUIs containing items of multiple taxonomy levels in *Liquor* such as $\{Cordials\&\ Liqueurs, Neutral\ Grain\ Spirits\}$ and $\{Liqueurs, Distilled\ Spirits\ Specialty\}$ that HUI-Miner and ML-HUI Miner cannot find.

6 Conclusion

This paper has defined a novel problem of mining cross-level HUIs, studied its properties and presented a novel algorithm named CLH-Miner to efficiently mine all cross-level HUIs. The algorithm integrates novel pruning strategies to reduce the search space by taking advantages of the relationships between abstraction levels. An extensive experimental evaluation has shown that the algorithm has excellent performance, that optimizations improves its performance, and that interesting patterns are found in real-life retail data. Source code and datasets of CLH-Miner are available in the SPMF data mining library [4]. In future work, we will generalize the proposed problem to consider using a hierarchy in other pattern mining problems such as episode mining [7, 8], subgraph mining [3], and high-average utility mining [24].

References

1. Cagliero, L., Cerquitelli, T., Garza, P., Grimaudo, L.: Misleading generalized itemset discovery. Expert Syst. Appl. **41**, 1400–1410 (2014)
2. Cagliero, L., Chiusano, S., Garza, P., Ricupero, G.: Discovering high-utility itemsets at multiple abstraction levels. In: Proc. 21st European Conference on Advances in Databases and Information Systems. pp. 224–234 (2017)
3. Fournier-Viger, P., Cheng, C., Lin, J.C.W., Yun, U., Kiran, R.U.: Tkg: Efficient mining of top-k frequent subgraphs. In: Proc. 7th International Conference on Big Data Analytics. pp. 209–226. Springer (2019)
4. Fournier-Viger, P., Lin, J.C.W., Gomariz, A., Gueniche, T., Soltani, A., Deng, Z., Lam, H.T.: The spmf open-source data mining library version 2. In: Proc. 20th European Conf. on machine learning and knowledge discovery in databases. pp. 36–40. Springer (2016)
5. Fournier-Viger, P., Lin, J.C.W., Truong-Chi, T., Nkambou, R.: A survey of high utility itemset mining. In: High-Utility Pattern Mining, pp. 1–45. Springer (2019)

6. Fournier-Viger, P., Lin, J.C.W., Vo, B., Chi, T.T., Zhang, J., Le, B.: A survey of itemset mining. WIREs Data Mining and Knowledge Discovery (2017)
7. Fournier-Viger, P., Wu, C.W., Zida, S., Tseng, V.S.: Fhm: Faster high-utility itemset mining using estimated utility co-occurrence pruning. In: Proc. 21st Intern. Symposium on Methodologies for Intelligent Systems. pp. 83–92 (2014)
8. Fournier-Viger, P., Yang, P., Lin, J.C.W., Yun, U.: Hue-span: Fast high utility episode mining. In: Proc. 14th Intern. Conf. on Advanced Data Mining and Applications. pp. 169–184. Springer (2019)
9. Fournier-Viger, P., Yang, Y., Yang, P., Lin, J.C.W., Yun, U.: Tke: Mining top-k frequent episodes. In: Proc. 33rd Intern. Conf. on Industrial, Engineering and Other Applications of Applied Intelligent Systems. Springer (2020)
10. Han, J., Fu, Y.: Mining multiple-level association rules in large databases. IEEE Trans. Knowl. Data Eng. **11**, 798–804 (1999)
11. Hashem, T., Ahmed, C.F., Samiullah, M., Akther, S., Jeong, B.S., Jeon, S.: An efficient approach for mining cross-level closed itemsets and minimal association rules using closed itemset lattices. Expert Syst. Appl. **41**, 2914–2938 (2014)
12. Hipp, J., Myka, A., Wirth, R., Güntzer, U.: A new algorithm for faster mining of generalized association rules. In: Proc. 2nd Pacific-Asia Conf. on Knowledge Discovery and Data Mining. pp. 74–82 (1998)
13. Liu, Y., keng Liao, W., Choudhary, A.N.: A two-phase algorithm for fast discovery of high utility itemsets. In: Proc. 9th Pacific-Asia Conf. on Knowledge Discovery and Data Mining. p. 689–695 (2005)
14. Lui, C.L., Chung, K.F.L.: Discovery of generalized association rules with multiple minimum supports. In: Proc. 4th European Conference Principles of Data Mining and Knowledge Discovery. pp. 510–515 (2000)
15. Luna, J.M., Fournier-Viger, P., Ventura, S.: Frequent itemset mining: A 25 years review. Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery **9**(6), e1329 (2019)
16. Mao, Y.X., Shi, B.L.: Afopt-tax: An efficient method for mining generalized frequent itemsets. In: Proc. 2nd Intern. Conf. on Intelligent Information and Database Systems. pp. 82–92 (2010)
17. leong Ong, K., Ng, W.K., Lim, E.P.: Mining multi-level rules with recurrent items using fp'-tree. In: Proc. 3rd Int. Conf. Inf. Comm. and Signal Processing (2001)
18. Pramudiono, I.: Fp-tax: Tree structure based generalized association rule mining. In: Proc. ACM/SIGMOD Intern. Workshop on Research Issues on Data Mining and Knowledge Discovery. pp. 60–63 (2004)
19. Qu, J.F., Liu, M., Fournier-Viger, P.: Efficient algorithms for high utility itemset mining without candidate generation. In: High-Utility Pattern Mining, pp. 131–160. Springer (2019)
20. Rajkumar, N.D., Karthik, M.R., Sivanandam, S.N.: Fast algorithm for mining multilevel association rules. Proc. 2003 TENCON Conference on Convergent Technologies for Asia-Pacific Region **2**, 688–692 (2003)
21. Srikant, R., Agrawal, R.: Mining generalized association rules. In: Proc. 21th Intern. Conf. on Very Large Data Bases (1995)
22. Sriphaew, K., Theeramunkong, T.: A new method for finding generalized frequent itemsets in generalized association rule mining. In: Proc. 7th Intern. Conf. Know. Based Intell. Inf. and Eng. Systems. pp. 476–484 (2002)
23. Xu, Y., Zeng, M., Liu, Q., Wang, X.: A genetic algorithm based multilevel association rules mining for big datasets. Mathematical Problems in Engineering (2014)
24. Yun, U., Kim, D., Yoon, E., Fujita, H.: Damped window based high average utility pattern mining over data streams. Knowledge-Based Systems **144**, 188–205 (2018)