This is the preprint of:

Fournier-Viger, P., Zhang, Y., Lin, J. C.W., Fujita, H., Koh, Y.S. (2019). Mining Local and Peak High Utility Itemsets. Information Sciences, Elsevier (to appear).

Source code and datasets available at : http://www.philippe-fournier-viger.com/spmf/

Mining Local and Peak High Utility Itemsets

Philippe Fournier-Viger[a,*], Yimin Zhang[b], Jerry Chun-Wei Lin[c], Hamido Fujita[d], Yun Sing Koh[e]

[a]*School of Humanities and Social Sciences, Harbin Institute of Technology (Shenzhen), Shenzhen, Guangdong, China, 518055*
[b]*School of Computer Sciences and Technology, Harbin Institute of Technology (Shenzhen), Shenzhen, Guangdong, China, 518055*
[c]*Department of Computing, Mathematics and Physics, Western Norway University of Applied Sciences (HVL), Bergen, Norway, 5020*
[d]*Iwate Prefectural University, Morioka, Japan, 020-8550*
[e]*Department of Computer Sciences, University of Auckland, Auckland, New Zealand, 303476*

Abstract

A major limitation of traditional High Utility Itemset Mining (HUIM) algorithms is that they do not consider that the utility of itemsets may vary over time. Thus, traditional HUIM algorithms cannot find itemsets that do not yield a high utility when considering the whole database, but still have a high utility during specific time periods. Discovering such itemsets is useful, as a product may sell exceptionally well during specific time periods but not during the rest of the year. This paper addresses this limitation of HUIM by defining the problem of mining *local high utility itemsets* (LHUI), and an extension to mine *peak high utility itemsets* (PHUI), which consists of finding the time periods where an itemset generates a utility that is much higher than usual (a peak). Algorithms named LHUI-Miner and PHUI-Miner are proposed to mine these patterns. Moreover, because the set of PHUIs can be large and some items in PHUIs don't contribute much to their peaks, a third algorithm named NPHUI-Miner is proposed to discover a smaller set of patterns called Non-redundant Peak High Utility Itemsets (NPHUIs). Experimental results show that the proposed algorithms are efficient and can find useful patterns.

Keywords: high-utility pattern mining, local high-utility itemsets, peak high-utility itemsets

1. Introduction

Association Rule Mining (ARM) [1] is a fundamental data mining task, which consists of discovering interesting associations between purchased items in a transaction database. The first step of ARM is called *Frequent Itemset Mining* (FIM). It consists of finding all sets of items that appear in at least *minsup* transactions of the database, where *minsup* is a user-defined threshold [1, 17]. FIM is a popular data mining task with many applications. However, it assumes that all items in a database are equally important and can appear at most once in each transaction. To address this limitation, *High-Utility Itemset Mining* (HUIM) [12, 19, 24, 25, 31, 35] has recently been considered as an important data mining task. It consists of finding itemsets (sets of items) that yield a high utility (e.g. profit or importance) in customer transaction databases. An itemset is a High Utility Itemset (HUI) in a database if its utility is no less than a user-specified minimum utility threshold. HUIM is widely viewed as a more difficult problem than FIM since the utility measure used in HUIM is not anti-monotonic, unlike the support measure used in FIM. In other words, the utility of an itemset may be greater, equal or smaller than the utility of its supersets. Thus, traditional FIM techniques cannot be directly used in HUIM to reduce the search space [1]. As a solution, HUIM algorithms such as Two-Phase [24] calculate upper-bounds on the utility measure, which are anti-monotonic, to reduce the search space.

Though HUIM has many applications such as click stream analysis, market basket analysis and biomedical applications [25, 31, 35], a major limitation of HUIM is that it ignores the time at which transactions were made. But considering the timestamps of transactions is important as the utility of patterns may vary over time. For example, *periodic high-utility itemset mining* [23] discovers itemsets that are periodically bought by customers and yield a high

*Corresponding author
Email addresses: philfv8@yahoo.com (Philippe Fournier-Viger), mrzhangym@126.com (Yimin Zhang), jerrylin@ieee.org (Jerry Chun-Wei Lin), hfujita-799@acm.org (Hamido Fujita), ykoh@cs.auckland.ac.nz (Yun Sing Koh)

profit. However, it does not consider the timestamps of transactions (only their relative order), and tend to find patterns that are stable in terms of utility in the whole database. Another related work is *High On-shelf Utility itemset Mining* (HOUM) [13], where each transaction is associated to a user-predefined time period (e.g. winter), and each item is associated to a set of periods indicating when it was sold. However, a major limitation of HOUM is that the utility of itemsets is calculated based on the predefined time periods (e.g. *winter*), which is unrealistic because many products may sell well during periods that do not match the predefined periods. Besides, HOUM also tend to find patterns that are stable in terms of utility in time periods where they are sold. Another related problem is to detect time points where the frequency of itemsets change significantly in data streams [37]. However, this problem also only consider the relative order of transactions instead of their real timestamps. In other words, this problem makes an unrealistic assumption that the time interval between any consecutive transactions is the same.

To find patterns that are profitable in non-predefined time periods, this paper proposes to discover a new type of patterns called *Local High Utility Itemsets* (LHUI). It consists of finding itemsets that yield a utility that is no less than a user-specified threshold during one or more time periods having a minimum time length. This allows to discover useful patterns such that the itemset $\{schoolbag, pen, notebook\}$ yields a high profit during the back-to-school shopping season, while not being a HUI in the whole database or in predefined time periods such as summer or autumn.

An efficient algorithm called LHUI-Miner is designed to discover LHUIs. It relies on a novel data structure named LU-list, and extends the basic search procedure and utility-list data structure of the HUI-Miner algorithm [25]. Besides, since the utilities of itemsets vary over time, it is desirable to find the time points where the utilities of itemsets change (increase or decrease) dramatically. Thus, the second major contribution of this paper is to extend the problem of mining LHUIs to mine *peak high utility itemsets* (PHUI). It consists of finding time periods where an itemset has a utility that is unusually high. Discovering PHUIs is desirable in market basket analysis for procurement and management, as it can identify itemsets and time periods where itemsets yield a profit that is higher than usual. An algorithm called PHUI-Miner is proposed to mine these itemsets. Lastly, as the set of PHUIs can be quite large and some items appearing in PHUIs don't contribute much to their peaks, a third problem is proposed, which is to mine a smaller set of patterns called the Non redundant Peak High Utility Itemsets (NPHUIs). An algorithm named NPHUI-Miner is designed for this problem.

The proposed algorithms can be viewed as providing more rich information to users than traditional HUI mining algorithms, as the proposed algorithms identify time intervals where a pattern has a high utility rather than just checking if a pattern has a high utility in the whole database. In fact, this paper demonstrates that the traditional problem of HUI mining is a special case of the problem of mining LHUIs.

The rest of this paper is organized as follows. Section 2 discusses related work. Section 3 introduces preliminaries and Section 4 defines the problems of mining LHUIs and PHUIs. Section 5 presents the proposed data structure and algorithms. Section 6 presents the experimental evaluation. Lastly, Section 7 draws the conclusion.

2. Related Work

This section surveys relevant related work on (1) frequent pattern mining and (2) high utility pattern mining.

2.1. Frequent itemset mining

Frequent itemset mining [1] is a fundamental data mining task. It consists of discovering sets of values that frequently co-occur in a database. Although, FIM has numerous aplications, it is usually described in the context of market basket analysis, where the goal is to find frequently purchased sets of items in a database of customer transactions. A frequent itemset is an itemset that appears in at least *minsup* transactions of a transaction database, where *minsup* is a parameter set by the user. Multiple FIM algorithms have been proposed. The first one, named Apriori [1], explores the search space of itemsets using a breadth-first search. It scans the database to calculate the frequency (support) of itemsets containing single items. Then, it recursively combines these itemsets to generate larger itemsets. The support of each generated candidate itemset is obtained by scanning the database, and only the frequent itemsets are shown to the user. To avoid exploring the whole search space of frequent itemsets, Apriori utilizes the anti-monotonicity property of frequent itemsets to reduce the search space, which states that supersets of an infrequent itemset are also infrequent [1]. Although Apriori guarantees finding all frequent itemsets, an important drawback of Apriori is that it repeatedly scans the database, which can result in long execution times.

Eclat [42] is another representative FIM algorithm, which is designed to address this drawback. The Eclat algorithm scans the database once to create a vertical structure for each item indicating the list of transactions where it appears. The vertical structure of an itemset allows calculating its support. Then, the vertical structure of an itemset containing two or more items can be obtained without scanning the database by joining the vertical structures of two of its subsets. Moreover, differently from Apriori, Eclat explores the search space of itemsets using a depth-first search.

Another popular FIM algorithm is FP-Growth [17]. It performs a depth-first search and repeatedly scans the database to calculate the support of itemsets. To reduce the cost of database scans, FP-Growth performs database projections and utilizes a compact tree-based database representation called FP-tree. Other algorithms that apply the concept of database projection include LCM [36] and H-Mine [30], which adopt horizontal and hyperstructure database representations, respectively. Besides Apriori, Eclat, FP-Growth, LCM and H-Mine, several other algorithms have been proposed in recent years to increase the performance of FIM [11]. Moreover, many other variations of the frequent pattern mining problem have been studied in recent years such as discovering itemsets in streams [7, 32, 41] episodes [27, 45], sequential patterns [10], association rules [1], high occupancy itemsets [34], and productive itemsets [8]. Several measures have been used to assess the interestingness of frequent patterns besides the support such as the bond [9], affinity [39], all-confidence [29], coherence and mean [5, 33].

Another related problem is to detect time points where the frequency of itemsets change significantly in data streams [37]. However, this approach does not consider the timestamps of transactions but only their relative order. In other words, this problem makes an unrealistic assumption that the time interval between any consecutive transactions is the same. Another problem of this approach is that to evaluate if a time point is interesting it divides the database into two parts (before and after the point) to compare if the frequency change significantly. Thus, it can't capture a short-term change. Lastly, as other traditional FIM approaches, it ignores the utility of itemsets.

2.2. High utility pattern mining

Although frequent itemset mining is useful, it has three important limitations. The first one is that purchase quantities of items in transactions are ignored. Thus, in FIM, buying several units of an item is considered as equally important as buying a single unit. This is unrealistic for aplications such as market basket analysis. The second limitation is that all items are considered as equally important. But in real life, some items may be more important to the user. For example, in a retail store, the sale of a laptop is viewed as more important than the sale of a computer since the former yields a much higher profit. The third limitation is that frequent patterns may not the most interesting to the user. For example, in market basket analysis, the amount of profit may be more important than the selling frequency.

To address these limitations, high utility itemset mining has recently emerged as an important research problem [12, 24, 35, 43]. It generalizes FIM by considering that items may have different utility, where the utility value of an item indicates its relative importance (e.g. weight or unit profit). Moreover, high utility itemset mining extends FIM by considering that items may appear more than once in each transaction. The goal of high utility itemset mining is to find all sets of items having a utility (e.g. profit) that is no less than a threshold set by the user. For instance, in the context of market basket analysis, it consists of finding all the itemsets that yield a profit that is at least equal to some minimum utility value. FIM is a special case of high utility itemset mining where all purchase quantities are binary and all items have the same utility.

From an algorithmic perspective, HUIM is widely considered as more difficult than FIM because the anti-monotonicity property used in FIM to reduce the search space does not hold for the utility measure. In other words, the utility of an itemset may be equal, greater or lower than that of its subsets. Two-Phase [24] is the first correct and complete algorithm for high utility itemset mining. It extends the Apriori algorithm and relies on an upper-bound on the utility, called TWU, to reduce the search space. Because high utility itemset mining is a difficult problem, several more efficient algorithms have been proposed. Inspired by the FP-Growth, algorithm, UP-Growth [35] compresses the input transaction database using a tree structure and performs database projections to reduce the cost of database scans. The UP-Growth algorithm relies on the TWU upper-bound to reduce the search space but also introduces several strategies to make this upper-bound tighter. As a result, UP-Growth obtain lower estimations of the utility of itemsets and hence can prune larger parts of the search space. However, a drawback of early high utility itemset mining algorithms such as Two-Phase and UP-Growth is that they often generate a large amount of candidate itemsets by overestimating their utility and that scanning the database to calculate their real utility is time-consuming.

To address limitations of these algorithms, several algorithms have been proposed that directly calculate the utility of candidates without scanning the database, and using tighter upper-bounds or upper-bounds that are easier to calculate. Some representative algorithms of this type are HUI-Miner [25], FHM [12], EFIM [43], d^{2}hup [26] and ULB-Miner [6]. The HUI-Miner [25] algorithm is inspired by the Eclat [42] algorithm. It construct a vertical structure called Utility-List for each itemset and uses this structure to calculate the utility and upper-bounds for any itemset. The FHM [25] algorithm is an improvement of HUI-Miner, which applies a pruning strategy based on the calculation of the TWU of pairs of items to reduce the search space. The ULB-Miner [6] algorithm improves the memory management of FHM [25] and HUI-Miner [25], by reusing the memory for storing utility-lists. It achieves better memory efficiency than FHM and HUI-Miner, and have smaller runtimes. The d^{2}hup [26] algorithm utilizes a hyper-structure database representation and perform database projections, similarly to H-Mine [30] and applies and upper-bound that is equivalent to the one of HUI-Miner [25]. The EFIM [43] algorithm applies an approach that is similar to LCM [36] by recursively creating projected horizontal databases and by merging identical transactions in projected databases. It was shown that EFIM can be several orders of magnitude faster than d^{2}hup, FHM, HUI-Miner and UP-Growth on dense datasets but does not always perform the best on sparse datasets.

Developing HUIM algorithms is an active research area, and new algorithms are regularly proposed. However, an important limitation of HUIM is that it does not consider the time at which transactions were made. A few extensions of HUIM have been designed to consider time. In High On-shelf Utility itemset Mining (HOUM) [13], each transaction is associated to a predefined time period (e.g. winter), and each item is associated to a set of periods indicating when it was sold. Then, HOUM consists of finding all itemsets that yield a high utility when they were on the shelves. However, a major limitation of HOUM is that the utility of itemsets is calculated based on the predefined time periods, which is unrealistic in most case because most products have different on-shelf time that may not match these predefined periods. A limitation of this approach is that timestamps are not considered but only the relative ordering of transactions, i.e. that the time interval between consecutive transactions is always the same. This limitation also exists in many other studies such as those about periodic HUIM [13, 23], which try to find itemsets that are periodically bought by customers but ignore the timestamps of transactions.

Few studies consider the timestamps of transactions. The problem of High Utility Episode Mining (HUEM) was proposed to discover subsequences of events (called episodes) having a high utility that appear in a complex event sequence [16, 22, 38]. A complex event sequence can be seen as a transaction database, where each transaction is a set of events (items) having a timestamp. Discovering high utility episodes is useful for discovering sequential relationships between events in a sequence, which is different from itemset mining that aims at finding patterns appearing in many transactions. For example, HUEM is useful to discover profitable sequences of purchases made by a customer in a retail store. To discover sequential relationships between events that have a high utility and appear in multiple sequences rather than a single sequence, the problem of High Utility Sequential Pattern Mining (HUSPM) was proposed [3, 4, 40]. Then, a variation of this problem called High Utility Sequential Rule Mining (HUSRM) was proposed to discover sequential association rules that are profitable and appear in multiple sequences [44].

Algorithms for HUEM, HUSPM and HUSRM are different from itemset mining algorithms as they must consider the sequential ordering between items. An important limitation of these algorithms is that although they consider the sequential ordering, they do not consider that trends may change over time, that is that some patterns may be more or less important at different times. For example, although a customer may perform a pattern many times during a time period, he may then stop doing it. Not considering that trends can change over time in a database can result in a bias towards finding patterns that have a utility that is more stable in the whole database, and can lead to ignore patterns that have a very high utility in some short time periods. Moreover, it is sometimes more interesting to find changing trends in the data rather than finding patterns that have a stable utility. For example, discovering that a pattern yield a very high profit during a specific time period of the year, can be used to promote that item during this period.

Other studies have considered timestamps of transactions, and that trends may change over time. Up-to-Date High Utility Pattern Mining algorithms were designed to find patterns that yield a high utility in recent transactions using a time-decay function to give more weight to recent transactions when calculating the utility of patterns [21]. A similar problem is Recent High Utility Itemset Mining [14], which uses a similar definition of recency. Moreover, the problem of finding recent high utility patterns in a stream was also studied [18]. A limitation of these studies is that they only focus on discovering recent patterns. However, not only recent changes in trends are interesting in a database. For example, when analyzing a year of customer transaction data, it may be interesting to find that a pattern has a high utility that is considerably higher than usual during a holiday, even though this pattern has not been recently

Table 1: A transaction database

Trans.	Items	Timestamp
T_1	$(b, 2), (c, 2), (e, 1)$	d_1
T_2	$(b, 4), (c, 3), (d, 2), (e, 1)$	d_3
T_3	$(b, 2), (c, 2), (d, 5), (e, 1)$	d_3
T_4	$(a, 2), (b, 10), (c, 2), (d, 10), (e, 2)$	d_5
T_5	$(a, 2), (c, 6), (e, 2)$	d_6
T_6	$(b, 4), (c, 3), (e, 1)$	d_7
T_7	$(a, 2), (c, 2), (d, 2)$	d_9
T_8	$(a, 2), (c, 6), (e, 2)$	d_{10}

Table 2: External utilities of items

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

profitable.

In this paper, the above limitations of HUIM are addressed by defining three new problems: mining local high utility itemsets, mining peak high utility itemsets, and mining non redundant peak high utility itemsets. The first problem aims at finding specific time intervals where a pattern has a high utility. The second problem aims at discovering time intervals where an itemset has a utility that is much higher than usual. The third problem is a variation of the second problem to eliminate some form of redundancy.

3. Preliminaries

This section introduces preliminaries related to the problem of high utility itemset mining [24, 35, 43]. Let $I = \{i_1, i_2, \ldots, i_n\}$ be a set of items. A transaction T is a subset of items purchased by a customer ($T \subseteq I$). A *transaction database* is a set of transactions $D = \{T_1, T_2, \ldots, T_m\}$, where each transaction T_{tid} ($1 \leq tid \leq m$) has a unique identifier *tid*. Moreover, let $t(T)$ denotes the time at which a transaction T was made. Each item $i \in I$ is associated with a positive number $p(i)$, called its external utility, which indicates its relative importance (e.g. unit profit). For each transaction T and item $i \in T$, a positive number $q(i, T)$ is called the internal utility of i, and represents the purchase quantity of i in T.

Example 1. *Consider the database of Table 1, which will be used as a running example. This database contains eight transactions ($T_1, T_2, \ldots, T_8$) and five items (a, b, c, d, e), where internal utilities (e.g. quantities) are shown as integers beside items. For instance, transaction T_1 indicates that 2, 2 and 1 units of items b, c and e were purchased, respectively. Table 2 indicates that the external utilities (unit profits) of these items are 2, 1 and 3. In this example, timestamps of transactions $T_1, T_2 ... T_8$ are $d_1, d_3, \ldots d_{10}$, representing days (d_i = i-th day), but other time units can be used such as milliseconds, and transactions can be simultaneous. Besides, although HUIM is presented in the context of market basket analysis for this example, it can be used for other applications [25, 35]*

Definition 1 (Utility of an itemset). The utility of an item i in a transaction T is defined as $u(i, T) = p(i) \times q(i, T)$. A set $X \subseteq I$ is an itemset. The utility of X in a transaction T is defined as $u(X, T) = \sum_{i \in X \wedge X \subseteq T} u(i, T)$. The utility of an itemset X in a database is defined as $u(X) = \sum_{T \in D \wedge X \subseteq T} u(X, T)$ *[24].*

Example 2. *For example, the utility of item b in T_1 is $u(b, T_1) = 2 \times 2 = 4$. The utility of itemset $\{b, c\}$ in T_1 is $u(\{b, c\}, T_1) = u(b, T_1) + u(c, T_1) = 2 \times 2 + 1 \times 2 = 6$. The utility of itemset $\{b, c\}$ in the database is $u(\{b, c\}) = u(\{b, c\}, T_1) + u(\{b, c\}, T_2) + u(\{b, c\}, T_4) + u(\{b, c\}, T_6) = 6 + 11 + 6 + 11 = 34$.*

Definition 2 (High utility itemset). An itemset X is said to be a *high utility itemset* (HUI) if its utility $u(X)$ is no less than a user-specified threshold *minutil* ≥ 0, that is $u(X) \geq$ *minutil [24].*

The HUIM problem generalizes the FIM problem. FIM is the special case of HUIM where all internal and external utility values in a database are set to either 0 or 1. HUIM is more difficult than FIM because the utility measure is not anti-monotonic (contrarily to FIM, where the support measure is anti-monotonic).

Property 1 (Utility is not monotonic nor anti-monotonic). Consider two itemsets X and Y such that $X \subset Y \subseteq I$. The utility $u(Y)$ of Y may be smaller, greater or equal to $u(X)$ [24].

Thus, a low utility itemset may have supersets that are low utility itemsets or high utility itemsets. This is why, search space reduction techniques used in FIM cannot be directly applied for mining HUIs. To address this challenge, the solution introduced in the Two-phase algorithm [24] has been to use and upper-bound on the utility, called Transaction Weighted Utilization (TWU), which is anti-monotonic.

Definition 3 (TWU upper-bound on the utility). Consider any transaction T. The transaction utility of T is denoted as $tu(T)$ and defined as $tu(T) = \sum_{i \in T} u(i, T)$. The TWU of an itemset X is the sum of utilities of transactions where it appears. Formally, $TWU(X) = \sum_{X \subseteq T \wedge T \in D} tu(T)$ [24].

Example 3. *The utility of transaction T_5 is $tu(T_5) = u(a, T_5) + u(c, T_5) + u(e, T_5) = 10 + 6 + 3 = 19$. The TWU of itemset $\{a, e\}$ in the database is $TWU(\{a, e\} = tu(T_5) + tu(T_8) = 19 + 19 = 38$.*

Several HUIM algorithms have used the following property of the TWU to reduce the search space [12, 25, 24].

Property 2. Consider two itemsets $Y \subset X \subseteq I$. Then, $TWU(Y) \geq TWU(X)$ and $TWU(X) \geq u(X)$ [24].

Property 3 (Search space pruning using the TWU). If $TWU(X) < minutil$ for any itemset X, it follows that $u(X) < minutil$ (X is not a high utility itemsets). Moreover, all supersets of X are low utility itemsets [24].

Example 4. *Consider that minutil $= 60$. The TWU of $\{a, e\}$ in the database is 38. Thus, $\{a, e\}$ and all its supersets are low utility itemsets.*

After the introduction of the TWU, several other upper-bounds on the utility have been proposed, which are tighter than the TWU. One of the most popular upper-bound is the remaining utility upper-bound, introduced in HUI-Miner [25]. Consider a total order $\prec$ on items in I. The remaining utility upper-bound is defined as follows.

Definition 4 (Remaining utility in a database). Consider an itemset X. Its remaining utility in a transaction T is calculated as $ru(X, T) = \begin{cases} \sum_{i \in T \wedge \forall j \in X, i \succ j} u(i, T) & \text{if } X \subseteq T \\ 0 & \text{otherwise} \end{cases}$. Then, the remaining utility of X is calculated as $ru(X) = \sum_{T \in D} ru(X, T)$ *[25].*

Definition 5 (Remaining utility upper-bound in a database). Let X be an itemset. The *remaining utility upper-bound* of X is defined as $reu(X) = u(X) + ru(X)$ [25].

Example 5. *Consider that the total order $\prec$ on items is defined as $a \prec b \prec c \prec d \prec e$, The remaining utility of itemset $\{b, c\}$ in transaction T_1 is $ru(\{b, c\}, T_2) = u(d, T_2) + u(e, T_2) = 4 + 3 = 7$. The remaining utility of itemset $\{b, c\}$ in the database is $ru(\{b, c\}) = ru(\{b, c\}, T_1) + ru(\{b, c\}, T_2) + ru(\{b, c\}, T_3) + ru(\{b, c\}, T_4) + ru(\{b, c\}, T_6) = 3+7+3+26+3 = 42$. The remaining utility upper-bound of itemset $\{b, c\}$ in the database is $reu(\{b, c\}) = u(\{b, c\}) + ru(\{b, c\}) = 52 + 42 = 94$.*

Based on that definition, algorithms such as HUI-Miner [25] and FHM [12] apply the following property to reduce the search space.

Definition 6 (Itemset extension). Consider an itemset X. An itemset Y is said to be an *extension* of X if $Y = X \cup \{j\} \wedge \forall i \in X, j \succ i$. An itemset Y is a *transitive extension* of an itemset X if $X \subset Y \wedge \forall j \in (Y - X) \wedge i \in X, j \succ i$.

Property 4 (Search space reduction using the remaining utility). If the remaining utility of an itemset X is less than *minutil* ($reu(X) < minutil$), then $u(X) < minutil$. Moreover, any transitive extension of X is also a low utility itemset [25].

Example 6. *Consider that minutil $= 60$. The remaining utility of $\{a, d\}$ is $reu(\{a, d\}) = u(\{a, d\}) + ru(\{a, d\}) = (30 + 14) + (3 + 0) = 47 < 60$. Hence, $u(\{a, d\}) < minutil = 60$ and any transitive extensions of $\{a, d\}$ is also a low utility itemset.*

4. Problem Statement

Though HUIM is useful, it is not designed to find patterns that describe how the utility of itemsets changes over time. This section proposes a solution to this limitation by defining the problems of identifying local high utility itemsets and peak high utility itemsets. The section then introduces an extension of the latter problem for mining non redundant peak high utility itemsets.

4.1. Local High Utility Itemset Mining

Traditional high utility itemset mining algorithms are designed to find itemsets that yield a high utility in a database rather than having a high utility in some time periods. But finding itemsets having a high utility in some time periods is useful. For instance, consider that $minutil = 60$. Because $u(\{d, e\}) = 46 < minutil$, $\{d, e\}$ is a low utility itemset. And since $u(\{a, c, e\}) = 62 > 60$, $\{a, c, e\}$ is a HUI. However, it can be oserved that $\{d, e\}$ has a utility that is more than twice that of $\{a, c, e\}$ from timestamp d_3 to d_5 (a utility of 46 compared to one of 18). But if a traditional high utility itemset mining algorithm is used to analyse the database, $\{d, e\}$ will not be found, although it is arguably more interesting than $\{a, c, e\}$ from d_3 to d_5. To find itemsets that are locally interesting in terms of the utility measure, this paper proposes to discover *Local High utility itemsets* (LHUIs). Those are patterns having a high utility in some time windows.

Definition 7 (Window). Let there be a database D containing m transactions, and two timestamps i, j (integers) such that $i \leq j$. The window from time i to j is denoted as $W_{i,j}$ and defined as $W_{i,j} = \{T | i \leq t(T) \leq j \wedge T \in D\}$. A window $W_{i,j}$ has a length $length(W_{i,j}) = j - i + 1$. The length W_D of a database D is calculated as $W_D = t(T_m) - t(T_1) + 1$. A window $W_{k,l}$ is said to subsume another window $W_{i,j}$ iff $W_{i,j} \subsetneq W_{k,l}$. Two windows $W_{i,j}$ and $W_{i+1,j+1}$ are called consecutive windows.

Example 7. *The window from time d_1 to d_3 is $W_{d_1,d_3} = \{T_1, T_2, T_3\}$. Its length is $length(W_{d_1,d_3}) = 3 - 1 + 1 = 3$. The windows W_{d_1,d_3} and W_{d_2,d_4} are consecutive windows. The window W_{d_1,d_3} subsumes W_{d_3,d_3}, while W_{d_3,d_3} is not subsumed by W_{d_2,d_3} since $W_{d_2,d_3} = \{T_2, T_3\} = W_{d_3,d_3}$.*

Definition 8 (Utility of an itemset in a window). Consider an itemset X and a window $W_{i,j}$. The utility of X in $W_{i,j}$ is calculated as $u_{i,j}(X) = \sum_{T \in W_{i,j} \wedge X \subseteq T} u(X, T)$.

Example 8. *In window W_{d_1,d_3}, the utility of $\{b, c\}$ is $u_{d_1,d_3}(\{b, c\}) = u(\{b, c\}, T_1) + u(\{b, c\}, T_2) + u(\{b, c\}, T_3) = 6 + 11 + 6 = 23$.*

To find patterns having a high utility in some time windows, while ensuring that these windows are not too small, a minimum window length parameter *minLength* is defined, representing a minimum time duration. This parameter is user-defined and must take a value in the $[1, W_D]$ interval. The *minLength* parameter is used to identify local high utility patterns, while not considering patterns that have a high utility in time periods smaller than *minLength.* For instance, if a user wants to find patterns having a high utility in a time period of at least one month, he would set $minLength = 1$ month. Besides *minLength*, another parameter named $lMinutil \geq 0$ is introduced. It lets a user specify the minimum utility that a pattern should have in a window to be considered interesting (e.g. profitable). If an itemset has a high utility in a window, then that window is called a *local high utility itemset period* of that itemset. Formally, it is defined as follows.

Definition 9 (LHUI period of an itemset). Consider an itemset X and a window $W_{i,j}$. The window $W_{i,j}$ is a *Local High Utility Itemset period (LHUI period)* of X if for any window $W_{k,l} \subseteq W_{i,j}$ where $length(W_{k,l}) = minLength$, $u_{k,l}(X) \geq lMinutil$.

Example 9. *Let there be $lMinutil = 30$ and $minLength = 5$. A LHUI period of $\{a, b, c\}$ is W_{d_2,d_7}. W_{d_2,d_6} and W_{d_3,d_7} are sub-windows of W_{d_2,d_7} having length 5. The utility of $\{a, b, c\}$ in the windows W_{d_2,d_6} and W_{d_3,d_7} are respectively $u_{d_2,d_6}(\{a, b, c\}) = u_{d_3,d_7}(\{a, b, c\}) = 32 > 30$.*

By the definition of LHUI period, any window $W_{k,l}$ of length *minLength* subsumed by a LHUI period $W_{i,j}$, must also be a LHUI period. The concept of LHUI period is defined in that way so that all *consecutive windows* where an itemset has a high utility are treated as a single window. Hence, multiple short windows that appear consecutively are replaced by a single large window. For instance, consider that $minLength = 2\ weeks$ and that an itemset has a high utility during five weeks. This period contains four consecutive windows of length two weeks that are LHUI periods for this itemset. To reduce the number of LHUI periods presented to the user by considering all these windows as a single large LHUI period, a concept of maximum LHUI period is introduced next. It consists of keeping a LHUI period only if it is not subsumed by another LHUI period.

Definition 10 (Maximum LHUI period). A LHUI period $W_{i,j}$ of an itemset is said to be a *maximum LHUI period* if there is no LHUI period $W_{o,p}$ such that $W_{i,j} \subset W_{o,p}$.

Example 10. *Consider that* $minLength = 5$ *and* $lMinutil = 30$. *The maximum LHUI period of* $\{a, b, c\}$ *is* W_{d_1,d_9}.

Another important consideration is that in a LHUI period of an itemset X, it is possible that X does not appear in the first and/or the last transactions of that period. For example, although the LHUI period of $\{a, b, c\}$ is $[d_1, d_9]$ in the above example, that itemset only appears in transaction T_4 of day d_5. To address this issue, the concept of abbreviated LHUI period of an itemset X is proposed, which consists of excluding the timestamps where X does not appear at the beginning and end of each LHUI period.

Definition 11 (Abbreviated LHUI period). Let there be a LHUI period $W_{i,j}$ of an itemset X. The abbreviated LHUI period $W_{k,l}$ representing $W_{i,j}$ is defined as the smallest window containing all the transactions from $W_{i,j}$ where X appears. Formally, it is the window $W_{k,l} \subseteq W_{i,j}$ such that $u_{i,j}(X) = u_{k,l}(X)$ and there does not exist another window $W_{w,z} \subset W_{k,l}$ such that $u_{w,z}(X) = u_{k,l}(X)$. That is k, l are the first and last timestamps of transactions in the LHUI period $W_{i,j}$ of X that contains X. In the following, the notation $X{:}[k, l]$ will be used to denote an abbreviated LHUI period $W_{k,l}$ of an itemset X.

Example 11. *Consider that* $minLength = 5$ *and* $lMinutil = 30$. W_{d_5,d_5} *is the abbreviated LHUI period of the LHUI period* W_{d_1,d_9} *of itemset* $\{a, b, c\}$.

Based on these concepts, the problem of *Local High Utility Itemset Mining (LHUIM)* is defined as follows.

Definition 12 (Local high utility itemset). An itemset X is a local high utility itemset (LHUI) in a database D if it has at least one LHUI period $W_{i,j}$.

Definition 13 (Local High Utility Itemset Mining). Let there be a database D and two parameters $minLength \leq W_D$ and $lMinutil > 0$. The problem of mining *Local High Utility Itemsets (LHUIM)* is to find all LHUIs and their abbreviated maximum LHUI periods. The set of all LHUIs is denoted as $LHUIs$.

Example 12. *Given the database of Table 1,* $minLength = 5$ *and* $lMinutil = 30$, *27 LHUIs are found with their abbreviated maximum LHUI periods, as illustrated in Table 3.*

The relationship between the set of HUIs discovered by traditional HUIM algorithms and the proposed set of LHUIs depends on how the parameters are set by the user. The following theorems explain this relationship.

Theorem 1. If $minLength = W_D$ and $lMinutil = minutil$, then $LHUIs = HUIs$.

Proof 1. An itemset X is a LHUI if it has at least one LHUI period, that is a window $W_{i,j}$ such that for each window $W_{k,l} \subseteq W_{i,j}$ of $length(W_{k,l}) = minLength$, $u_{k,l}(X) \geq lMinutil$. Since $minLength = W_D$, X is a LHUI if $u_{o,p}(X) \geq lMinutil$ where o and p are the first and last timestamp of the database, respectively. This is equivalent to $u(X) \geq minutil$. □

Theorem 2. If $minutil \geq lMinutil \times \lceil \frac{W_D}{minLength} \rceil$, then $HUIs \subseteq LHUIs$.

Table 3: Local high utility itemsets for *minLength* = 5 and *lMinutil* = 30, where high utility itemsets are highlighted in bold

LHUI	Utility	LHUI	Utility	LHUI	Utility
$\{a,b,c\}$:$[d_5,d_5]$	32	$\{a,b,c,d\}$:$[d_5,d_5]$	52	$\{a,b,c,d,e\}$:$[d_5,d_5]$	58
$\{a,b,c,e\}$:$[d_5,d_5]$	38	$\{a,b,d\}$:$[d_5,d_5]$	50	$\{a,b,d,e\}$:$[d_5,d_5]$	56
$\{a,b,e\}$:$[d_5,d_5]$	36	$\{a,c\}$:$[d_5,d_{10}]$	56	$\{a,c,d\}$:$[d_5,d_9]$	48
$\{a,c,d,e\}$:$[d_5,d_5]$	38	**{a,c,e}**:$[d_5,d_{10}]$	62	$\{a,d\}$:$[d_5,d_9]$	44
$\{a,e\}$:$[d_5,d_{10}]$	48	$\{b\}$:$[d_1,d_7]$	44	$\{b,c\}$:$[d_1,d_7]$	56
{b,c,d}:$[d_3,d_5]$	73	**{b,c,d,e}**:$[d_3,d_5]$	85	**{b,c,e}**:$[d_1,d_7]$	74
{b,d}:$[d_3,d_5]$	66	**{b,d,e}**:$[d_3,d_5]$	78	**{b,e}**:$[d_1,d_7]$	62
$\{c,d\}$:$[d_3,d_5]$	47	$\{c,d,e\}$:$[d_3,d_5]$	53	$\{c,e\}$:$[d_3,d_7]$	54
$\{d\}$:$[d_3,d_5]$	38	$\{d,e\}$:$[d_3,d_5]$	46	$\{a,d,e\}$:$[d_5,d_5]$	36

Proof 2. If Theorem 3 is false, then there exists an itemset $X \in HUIs$ such that $X \notin LHUIs$. This implies that for all window $W_{i,j}$ such that $length(W_{i,j}) = minLength, u_{i,j}(X) < lMinutil$. Assume that we divide the database into $\lceil \frac{W_D}{minLength} \rceil$ non overlapping windows having a length *minLength* (note that there may be a window of length less than *minLength* if $\frac{W_D}{minLength}$ is not an integer). Let WU_i denotes the utility of X in the i-th window. Since $\forall WU_i, WU_i < lMinutil$, it follows that $u(X) = \sum_{i=0}^{\lceil \frac{W_D}{minLength} \rceil} WU_i < lMinutil \times \lceil \frac{W_D}{minLength} \rceil \leq minutil$. There is a contradiction between $u(X) < minutil$ and $X \in HUIs$. Thus, Theorem 3 is proven. □

Example 13. *Given the database of Table 1, minLength = 5 and lMinutil = 30, 27 LHUIs are found with their maximum LHUI periods, as illustrated in Table 3. For minutil* $= lMinutil \times \lceil \frac{W_D}{minLength} \rceil = 60$, *traditional HUIM algorithms find seven HUIs:* $\{b,e\}$, $\{a,c,e\}$, $\{b,d,e\}$, $\{b,c,d,e\}$ *and* $\{b,c,e\}$. *In this example, all HUIs are LHUIs. HUIs are thus highlighted in bold in Table 3.*

Theorem 3. If $minutil < lMinutil \times \lceil \frac{W_D}{minLength} \rceil$, then one of the three following relationships hold (1) $LHUIs \subset HUIs$, (2) $HUIs \subset LHUIs$, or (3) $HUIs = LHUIs$.

Proof 3. The proof is made in three parts:
(1) To prove that the first case is possible, we can consider that $minutil = 0$ and $lMinutil = \infty$. In that case, the set of HUIs is the powerset of I, while the set of LHUIs is empty, and thus the first relationship holds.
(2) To prove that the second case is possible, consider the LHUIs and HUIs found in the database of Table 1 for $minLength = 5$ and $lMinutil = 30$. In that case, 27 LHUIs are found, as illustrated in Table 3, and seven HUIs, which are also LHUIs (highlighted in bold in Table 3). Thus the second relationship holds.
(3) To prove that the third case is possible, we can consider that *minutil* and *lMinutil* are both set to utility values that are greater than the utility of the database. In that case, $HUIs = LHUIs = \emptyset$ and the third relationship holds. □

4.2. Peak High Utility Itemset Mining

The previous section has proposed the problem of mining local utility itemsets. This section defines a second type of patterns based on LHUIs named *peak high utility itemsets* (PHUIs) to find itemsets that not only yield a high utility in some time periods but a utility that is also considerably higher than usual. Finding such patterns is desirable in many real-life applications. For example, in a retail store, identifying the time periods where an itemset yields a profit that is higher than usual can be used to improve procurement planning and management. The concept of *peak high utility itemsets* is based on the concept of moving average crossover used in time series analysis.

A time series ρ is an ordered list of numbers $\rho = \rho_1, \rho_2, \dots \rho_k$ that are assumed to be observations made at equally spaced points in time. Given a smoothing parameter n, the *moving average* of the i-th data point in a time series ρ is the mean of the previous n data points, that is $ma(\rho, i) = avg(\rho_{i-n} \dots \rho_{i-2}, \rho_{i-1})$. The moving average is used to smooth out short-term fluctuations in time series [2]. The larger n is, the more smoothing is obtained. In some studies, the central moving average is used instead of the moving average to not just consider data occurring before a data point but also after. Given a smoothing parameter n, the *central moving average* of the i-th data point in a time series ρ is the mean of the n data points that are the closest to i. Thus, the central moving average is calculated by considering an equal number of points before and after the i-th point if n is even.

A *moving average crossover* is a time point where two moving averages based on different degrees of smoothing cross each other. Detecting moving average crossovers is often used for stock trading [28]. In that context, crossovers are interpreted as signals for buying/selling stocks. For example, consider Fig. 1, which depicts a time series and its moving averages for $n = 5$ and $n = 10$ (called short and long moving averages, respectively), and where the moving average crossovers are indicated with arrows. The crossovers can be interpreted as follows. If the short moving average crosses above the long moving average, it indicates an upward trend, while if it crosses below, it indicates a downward trend.

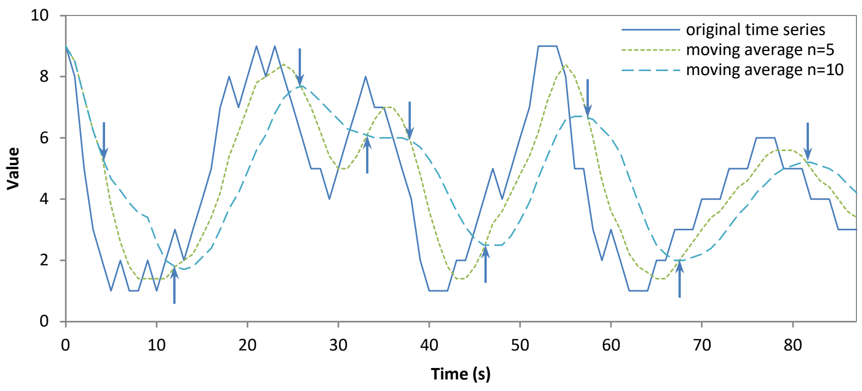

Figure 1: Crossovers of the short and long moving averages of a time series

In the following, the concept of moving average crossover is adapted to find the start and end of time intervals where the utility of itemsets is much higher than usuaul. But unlike the traditional moving average crossover, the central moving average is used instead of the moving average. The reason is that we assume that a transaction database is not analyzed in real-time to discover patterns and thus that the data immediately before and after a time point is available rather than only the previous data. This is different from other applications such as stock market analysis, where one may try to predict the future based on the past. Moreover, unlike traditional time series analysis, this paper adapts the central moving average to consider timestamps, to avoid relying on the assumption that data points are equally spaced in time. The modified moving average used in this paper is defined as follows.

Definition 14 (Moving average utility). Let there be an itemset X and a smoothing parameter $\gamma \geq 1$ representing a time length. The moving average utility of X for a timestamp t is defined as $mau_{\gamma}(X,t) = \frac{u_{t-\frac{\gamma-1}{2},t+\frac{\gamma-1}{2}}(X)}{\gamma}$, that is the average utility of X before and after t in a window of length γ.

Based on the concept of moving average utility, crossovers can be identified. However, this requires to define two moving averages, having a long and short window, respectively. This is defined as follows. Consider an itemset X and a timestamp t. The short term moving average utility of X at time t is denoted as $mau_{\gamma}(X,t)$ and is calculated using the window $W_{t-\frac{\gamma-1}{2},t+\frac{\gamma-1}{2}}$. Fig. 2 a) depicts that windows, which has a length of γ time units. Similarly, the long term moving average utility of X is denoted as $mau_{\lambda\times\gamma}(X,t)$ and is calculated over a window $W_{t-\frac{\lambda\times\gamma-1}{2},t+\frac{\lambda\times\gamma-1}{2}}$ where $\lambda \geq 1$ is a user-defined smoothing parameter. Fig. 2 b) depicts this window, which has a length of $\lambda \times \gamma$ time units. Based on these short and long windows, a concept of *peak window* is proposed. It is a time period where the utility of an itemset is much higher than usual.

Definition 15 (Peak window). For a LHUI X, a window $W_{i,j}$ is a peak window if $W_{i,j}$ is a LHUI period of X and $\forall i \leq t \leq j$, $mau_{minLength}(X,t) \geq mau_{\lambda\times minLength}(X,t)$, where λ is a user-defined parameter ($\lambda \geq 1$), called the moving average crossover coefficient. If $\lambda = 1$, the peak windows of X are its LHUI periods and if $\lambda = \infty$, the peak windows of X are the periods where its utility is greater than the average in the whole database.

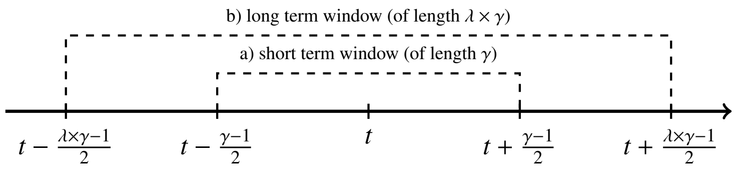
b) long term window (of length $\lambda \times \gamma$)
a) short term window (of length γ)
$t - \frac{\lambda\times\gamma-1}{2}$
$t - \frac{\gamma-1}{2}$
t
$t + \frac{\gamma-1}{2}$
$t + \frac{\lambda\times\gamma-1}{2}$

Figure 2: The windows for calculating the a) short term and b) long term moving average utility for a timestamp t.

Table 4: Peak high utility itemsets for $minLength = 3$, $lMinutil = 30$ and $lambda = 2$, where NPHUIs are highlighted in bold

PHUI	peak window	PHUI	peak window	PHUI	peak window
$\{a, b, c\}$	$[d_5, d_5]$	$\{a, b, c, d\}$	$[d_5, d_5]$	$\{a, b, c, d, e\}$	$[d_5, d_5]$
$\{a, b, c, e\}$	$[d_5, d_5]$	$\{a, b, d\}$	$[d_5, d_5]$	$\{a, b, d, e\}$	$[d_5, d_5]$
$\{a, b, e\}$	$[d_5, d_5]$	$\{a, c, d\}$	$[d_5, d_5]$	$\{a, c, d, e\}$	$[d_5, d_5]$
$\{a, c, e\}$	$[d_5, d_6]$	$\{a, d, e\}$	$[d_5, d_5]$	$\{a, e\}$	$[d_5, d_6]$
{b}	$[d_3, d_5]$	$\{b, c, d\}$	$[d_3, d_5]$	$\{b, c, d, e\}$	$[d_3, d_5]$
{b,d}	$[d_3, d_5]$	$\{b, d, e\}$	$[d_3, d_5]$	$\{c, d\}$	$[d_3, d_5]$
$\{c, d, e\}$	$[d_3, d_5]$	**{d}**	$[d_3, d_5]$	$\{d, e\}$	$[d_3, d_5]$

Example 14. *Consider that minLength* $= 3$, *lMinutil* $= 10$ *and* $\lambda = 2$. W_{d_5,d_6} *is a peak window of itemset* $\{a, c\}$. *Because* W_{d_5,d_6} *is a LHUI period of* $\{a, c\}$ *(*$u_{d_5,d_6}(\{a, c\}) = 28 > lMinutil$*) and* $mau_3(\{a, c\}, d_5) = mau_3(\{a, c\}, d_6) = \frac{28}{3} > mau_{3\times2}(\{a, c\}, d_5) = mau_{3\times2}(\{a, c\}, d_6) = \frac{28}{5}$.

The problem of *Peak High Utility Itemset Mining (PHUIM)* is defined as follows.

Definition 16 (Peak High Utility Itemset). An itemset X is a peak high utility itemset (PHUI) in a database if it has at least one peak window.

Definition 17 (Peak High Utility Itemset Mining). The problem of mining *Peak High Utility Itemsets (PHUIM)* in a database D is to find all the peak high utility itemsets with their peak windows, given user-defined parameters $1 \leq minLength \leq W_D$, $lMinutil > 0$ and $\lambda > 1$. The set of all PHUIs is denoted as $PHUIs$.

Example 15. *Given the database of Table 1, minLength* $= 3$, *lMinutil* $= 30$ *and* $\lambda = 2$, *21 Peak High Utility Itemsets are found, as illustrated in Table 4.*

The relationship between the proposed set of PHUIs and LHUIs is the following.

Theorem 4. $PHUIs \subseteq LHUIs$. Moreover, in the case where $\lambda = 1$, $PHUIs = LHUIs$.

Proof 4. By definition, a PHUI is a LHUI, and thus $PHUIs \subseteq LHUIs$. Consider any LHUI period of a LHUI X. If $\lambda = 1$, any timestamps t in that LHUI period will satisfy the condition $mau_{minLength}(X, t) \geq mau_{\lambda\times minLength}(X, t)$ since $mau_{minLength}(X, t) = mau_{1\times minLength}(X, t)$. Hence, all LHUI periods of X are peak windows, X is a PHUI, and thus $PHUIs = LHUIs$.

4.3. Non Redundant Peak High Utility Itemset Mining

Discovering PHUIs is desirable because it can find time periods where itemsets yield a utility that is considerably higher than usual. However, a problem is that the number of PHUIs and their peak windows can be very large, as it will be shown in the experiment. It can be observed that some PHUIs contains items that do not contribute much to their peak windows. For example, consider Fig. 3 illustrating the average utility for two units of time (mau_2) of the itemset $\{c, d\}$ and its subsets $\{c\}$ and $\{d\}$, based on the database of Table 1. Although $\{c, d\}$ is a PHUI, the item c does not contribute much to its peak windows, and as a result the itemset $\{c, d\}$ is not interesting. To find fewer but more relevant PHUIs by eliminating all such itemsets, the concept of non redundant peak window and non redundant PHUI is proposed.

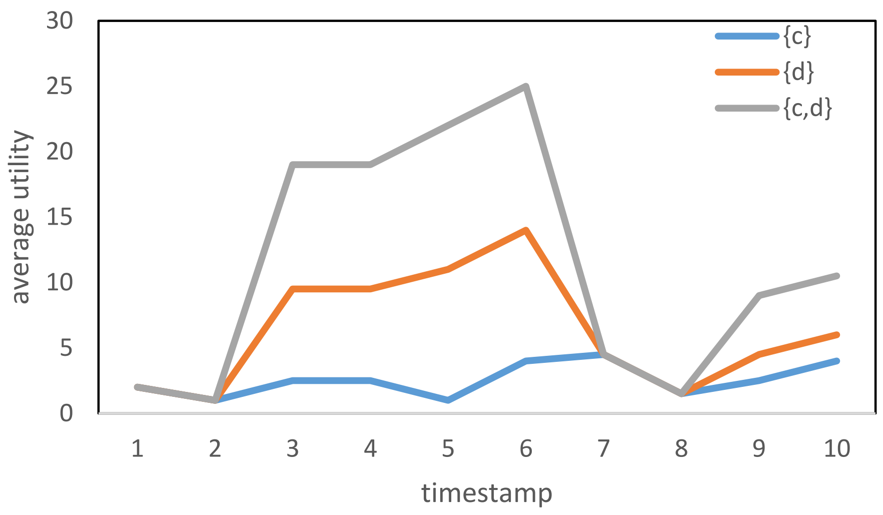

Figure 3: The average utility distributions of itemset $\{c, d\}$ and its subsets

Definition 18 (Non Redundant Peak window). A peak window $W_{i,j}$ of an itemset X is a non redundant peak window if $W_{i,j}$ is a peak window for each non empty subset $Y \subset X$.

Example 16. *Consider that* $minLength = 3$, $lMinutil = 30$ *and* $\lambda = 2$. W_{d_3,d_5} *is a non redundant peak window of itemset* $\{b, d\}$, *because* $\{b\}$ *and* $\{d\}$ *both have the peak window* $[d_3, d_5]$.

Definition 19 (Non Redundant Peak High Utility Itemset). A peak high utility itemset X is a Non Redundant Peak High Utility Itemset (NPHUI) if it has a least one non redundant peak window.

Definition 20 (Non Redundant Peak High Utility Itemset Mining). The problem of mining *Non redundant Peak High Utility Itemsets (NPHUIM)* in a database D is to find all the non redundant peak high utility itemsets with their peak windows, given user-defined parameters $1 \leq minLength \leq W_D$, $lMinutil > 0$ and $\lambda > 1$. The set of all NPHUIs is denoted as $NPHUIs$.

Example 17. *Given the database of Table 1,* $minLength = 3$, $lMintuil = 30$ *and* $\lambda = 2$*, 21 PHUIs are found as illustrated in Table 4, while only 3 NPHUIs are found, which are highlighted in bold in Table 4.*

The set of NPHUIs is a subset of the set of PHUIs, which is a subset of the set of LHUIs. NPHUIs have the following property, which directly follows from the definition of NPHUI.

Property 5 (anti-monotonocity for NPHUIs). If an itemset is not a NPHUI, then none of its supersets are NPHUIs.

Theorem 5. If there are less than $i + 1$ NPHUIs of any length i, then there exsits no NPHUIs containing more than i items.

Proof 5. Assume that there exists a NPHUI X containing $k > i$ items. According to Property 5, all subsets of X of length i must also be NPHUIs. The number of such subsets is $\binom{k}{i} \geq i + 1$. Thus, there is a contradiction and X cannot be a NPHUI. $\square$

5. Proposed Algorithms

This section presents three algorithm to efficiently mine LHUIs, PHUIs and NPHUIs, named LHUI-Miner, PHUI-Miner and NPHUI-Miner, respectively. The first subsection introduces two novel upper-bounds on the utility of itemsets in a window and two corresponding theorems to reduce the search space. The second subsection presents a novel data structure named LU-list, used by the proposed algorithms. Then, the following subsections present each algorithm, optimizations, and discuss their complexity.

5.1. *Two Novel Upper-bounds on the Utility of Itemsets in a Window*

The search space for the problems of mining LHUIs, PHUIs and NPHUIs in a database containing n items consists of $2^{|I|} - 1$ itemsets. Exploring the whole search space is thus impractical for real-life databases. It is thus necessary to design strategies to reduce the search space. In previous work, HUIM algorithms have mainly relied on two upper-bounds called the TWU and remaining utility upper-bound to reduce the search space (see Section 3). Although these upper-bounds were used to effectively prune the search space in HUIM, they cannot be directly utilized for the three problems studied in this paper, as they do not consider windows. This section adapts these upper-bounds to obtain two novel upper-bounds that consider the utility of itemsets in windows, and propose novel theorems to reduce the search space using these upper-bounds.

The TWU upper-bound is adapted as follows to consider the utility of an itemset in a window.

Definition 21 (TWU of an itemset in a window). The TWU of an itemset X in a window $W_{k,l}$ is defined as $TWU_{k,l}(X) = \sum_{X \subseteq T \wedge T \in W_{k,l}} tu(T)$. In the following, for an item i, the notation $TWU_{k,l}(i)$ will be used to refer to $TWU_{k,l}(\{i\})$.

Based on the designed TWU upper-bound of an itemset in a window, the following lemma and theorem are proposed. They are used by the designed algorithms to discard itemsets that cannot be LHUI, PHUI, or NPHUI, from the search space.

Lemma 1. Let there be an itemset X, an itemset $Y \supseteq X$ and a window $W_{k,l}$ such that $length(W_{k,l}) = minLength$. It follows that $TWU_{k,l}(X) \geq u_{k,l}(Y)$.

Proof 6.

$$X \subseteq Y \Rightarrow \{T | T \in W_{k,l} \wedge Y \subseteq T\} \subseteq \{T | T \in W_{k,l} \wedge X \subseteq T\} \qquad (1)$$

$$\begin{aligned} u_{k,l}(Y) &= \sum_{T \in W_{k,l} \wedge Y \subseteq T} u(Y,T) \\ &\leq \sum_{T \in W_{k,l} \wedge Y \subseteq T} tu(T) && \text{(because } tu(T) \geq u(Y,T)\text{)} \\ &\leq \sum_{T \in W_{k,l} \wedge X \subseteq T} tu(T) && \text{(because of (1))} \\ &= TWU_{k,l}(X) \square \end{aligned}$$

Theorem 6. For an itemset X, if for any window $W_{k,l}$ of length *minLength*, $TWU_{k,l}(X) < lMinUtil$, then no supsersets of X are LHUI, PHUI or NPHUI.

Proof 7. Let there be an itemset $Y \supseteq X$. For any window $W_{k,l}$ of length *minLength*, $TWU_{k,l}(X) < lMinUtil$. By Lemma 1, $TWU_{k,l}(X) \geq u_{k,l}(Y)$. Hence, $u_{k,l}(Y) \leq lMinUtil$. Thus, Y has no LHUI period, and is not a LHUI. Because $NPHUIS \subseteq PHUIs \subseteq LHUIs$ (by Theorem 4), then Y is also neither a PHUI or NPHUI. □

There exists a relationship between the TWU in a window and the TWU in a database, that allows to reduce the search space.

Corollary 1. For an itemset X, the relationship $TWU(X) \geq TWU_{k,l}(X)$ holds for any window $W_{k,l}$. Moreover, if $TWU(X) < lMinUtil$, then no supsersets of X are LHUI, PHUI or NPHUI.

Proof 8.
$\because W_{k,l} \subseteq D$
$\therefore TWU_{k,l}(X) = \sum_{T \in W_{k,l} \wedge X \subseteq T} tu(T) \leq \sum_{T \in D \wedge X \subseteq T} tu(T) = TWU(X)$
Therefore, if $TWU(X) < lMinutil$, $TWU_{k,l}(X) < lMinutil$ holds for any window $W_{k,l}$, and by Theorem 6, all supsersets of X are not LHUI, PHUI or NPHUI. □

A second upper-bound is proposed by adapting the remaining utility upper-bound to consider the utility of an itemset in a window.

Definition 22 (Remaining utility in a window). The remaining utility of an itemset X in a window $W_{k,l}$ is defined as $ru_{k,l}(X) = \sum_{T\in W_{k,l}} ru(X,T)$.

Definition 23 (Remaining utility upper-bound in a window). Let there be an itemset X and a window $W_{k,l}$. The remaining utility upper-bound of X is defined as $reu_{k,l}(X) = u_{k,l}(X) + ru_{k,l}(X)$.

The proposed remaining utility upper-bound of an itemset in a window is the basis of the following theorems used by the proposed algorithms to discard itemsets that cannot be LHUI, PHUI, or NPHUI.

Definition 24 (PLHUI period). For an itemset X, a window $W_{k,l}$ is a *PLHUI period* (promising LHUI period) if $\forall W_{y,z} \subseteq W_{k,l} \wedge length(W_{y,z}) = minLength$, $reu_{y,z}(X) \geq lMinutil$.

Example 18. *Consider that $a \prec b \prec c \prec d \prec e$, $minLength = 5$ and $lMinutil = 30$. The PLHUI-periods of $\{b,c\}$ is W_{d_1,d_7} because $u_{d_1,d_5}(\{b,c\}) = 45 > 30$, $u_{d_2,d_6}(\{b,c\}) = 38 > 30$ and $u_{d_3,d_7}(\{b,c\}) = 49 > 30$.*

Theorem 7. If a window $W_{k,l}$, such that $length(W_{k,l}) = minLength$, is not a PLHUI period of an itemset X, then it is also not a LHUI period for any transitive extension Y of X.

Proof 9. For two itemsets $W \subseteq Z$, let $E(W,Z) = \{j | j \in Z \wedge \forall i \in W, j > i\}$.
For all transaction $T \supseteq Y$:

$$\begin{aligned}
\because Y \text{ is a transitive extension of } X \Rightarrow Y - X &= E(X,Y) \\
X \subseteq T &\Rightarrow E(X,Y) \subseteq E(X,T) \qquad (1)\\
\therefore \text{ when } X \subseteq T,\ u(Y,T) &= u(X,T) + u((Y-X),T) \\
&= u(X,T) + u(E(X,Y),T) \\
&= u(X,T) + \sum_{i\in E(X,Y)} u(i,T) \\
&\leq u(X,T) + \sum_{i\in E(X,T)} u(i,T) \qquad \text{(because of (1))}\\
&= u(X,T) + \sum_{i\in T \wedge \forall j\in X, i>j} u(i,T) \\
&= u(X,T) + ru(X,T) \qquad (2)
\end{aligned}$$

Moreover,

$$\begin{aligned}
\because Y \text{ is a transitive extension of } X &\Rightarrow X \subset Y \\
&\Rightarrow \{T | T \in W_{k,l} \wedge Y \subseteq T\} \subseteq \{T | T \in W_{k,l} \wedge X \subseteq T\} \qquad (3)\\
\therefore u_{k,l}(Y) &= \sum_{T\in W_{k,l} \wedge Y\subseteq T} u(Y,T) \\
&\leq \sum_{T\in W_{k,l} \wedge Y\subseteq T} u(X,T) + ru(X,T) \qquad \text{(because of (2))}\\
&\leq \sum_{T\in W_{k,l} \wedge X\subseteq T} u(X,T) + ru(X,T) \qquad \text{(because of (3))}\\
&< lMinutil \qquad \text{(because } W_{k,l} \text{ is not a PLHUI period of } X)
\end{aligned}$$

Therefore, $W_{k,l}$ is not a LHUI period of itemset Y. □

Theorem 8. If an itemset X has no PLHUI period, then any transitive extension Y of X is neither a LHUI, PHUI or NPHUI.

Proof 10. Since X has no PLHUI period, then Y has no LHUI period according to Theorem 7. Hence, Y is not a LHUI. Because $NPHUIS \subseteq PHUIs \subseteq LHUIs$ (by Theorem 4), then Y is neither a LHUI, PHUI or NPHUI.

LU-list of {b}

Utility-list		
tid	*iutil*	*rutil*
T_1	4	5
T_2	8	10
T_3	4	15
T_4	20	28
T_6	8	6

iutilPeriods
$[d_1, d_7]$

utilPeriods
$[d_1, d_7]$

LU-list of {c}

Utility-list		
tid	*iutil*	*rutil*
T_1	2	3
T_2	3	7
T_3	2	13
T_4	2	26
T_5	6	6
T_6	3	3
T_7	2	4
T_8	6	6

iutilPeriods
$\emptyset$

utilPeriods
$[d_1, d_{10}]$

LU-list of {b,c}

Utility-list		
tid	*iutil*	*rutil*
T_1	6	3
T_2	11	7
T_3	7	13
T_4	22	26
T_6	11	3

iutilPeriods
$[d_1, d_7]$

utilPeriods
$[d_1, d_7]$

Figure 4: The LU-lists of itemsets $\{b\}, \{c\}$ and $\{b, c\}$

5.2. The LU-list data structure

The proposed algorithms extend the basic search procedure of the HUI-Miner [25] algorithm. This search procedure performs a depth-first search. The proposed algorithms explore the search space of itemsets by following a total order $\succ$ on items in I. In the implementation, the $\succ$ order is defined as the order of increasing TWU values since using that order can reduce the search space of HUIM [12, 25, 35]. However, we next consider that $\succ$ is the lexicographical order, to make the examples easier to understand for the reader.

The proposed algorithms utilizes a novel data structure called *Local Utility-list* (LU-list) to store information about each itemset, which extends the utility-list [25] structure to store additional information about periods. The algorithms first scan the database to create a LU-list for each item. Then, they explore the search space of itemsets using a depth-first search, by combining pairs of itemsets to generate their extensions and their LU-lists. A LU-list allows to determine if an itemset is a LHUI or PHUI without scanning the database. The LU-list structure is defined as follows by extending the utility-list structure.

Definition 25 (Utility-list). Let $\succ$ be any total order on I. The *LU-list* of an itemset X contains a tuple for each transaction that contains X. A *tuple* (also called *element*) has the form (*tid*, *iutil*, *rutil*), where *tid* is the identifier of a transaction T_{tid} containing X, *iutil* is the utility of X in T_{tid}. i.e. $u(X, T_{tid})$, and *rutil* is defined as $\sum_{i \in T_{tid} \wedge \forall j \in X, i \succ j} u(i, T_{tid})$ *[25].*

Definition 26 (LU-list). The *LU-list* of an itemset X is a utility-list with two additional sets named *iutilPeriods* and *utilPeriods*, which stores the abbreviated maximum LHUI periods and PLHUI periods of X, respectively.

Example 19. *Consider that $a \prec b \prec c \prec d \prec e$, minLength $= 3$ and lMinutil $= 30$. The LU-list of itemsets $\{b\}, \{c\}$ and $\{b, c\}$ are illustrated in Fig. 4.*

The LU-lists of an itemset stores information that can be used to directly obtain the utility of the itemset in any window, without scanning the database.

Property 6. If *iutilPeriods* in the LU-list of an itemset X is not empty, then X is a LHUI.

Proof 11. By definition, *iutilPeriods* contains the LHUI periods of X, and if an itemset has at least one LHUI period, then it is a LHUI.

Besides, the LU-list of an itemset can be used to directly obtain the remaining utility upper-bound of the itemset in a window (without scanning the database). And this can be used to reduce the search space.

Property 7. If *utilPeriods* in the LU-list of an itemset X is empty, then all its transitive extensions cannot be LHUIs or PHUIs and can be pruned from the search space.

Proof 12. This directly follows from Theorem 8.

The LU-list structure is thus useful to directly obtain the utility of itemsets and reduce the search space. The LU-lists are constructed as follows. The proposed algorithms build the LU-lists of single items by scanning the database once. The LU-list of itemsets containing more than one item are built by performing a join operation using the LU-lists of smaller itemsets. Consider an itemset P and two items x and y. Let the notation Px denotes the itemset $P \cup \{x\}$. The LU-list of an itemset Pxy is constructed in two steps. First, the *Construct* procedure of HUI-Miner [25] (Algorithm 1) is called with the LU-lists of P, Px and Py as parameters to create the utility-list of Pxy. Because this procedure is the same as HUI-Miner, the reader is referred to [25] for an explanation of this procedure. Then, the *iutilperiods* and *utilperiods* of the LU-list of Pxy are calculated using the *GeneratePeriods* procedure (Algorithm 2).

Algorithm 1: The Construct procedure

```
Input: P: an itemset, Px: the extension of P with an item x, Py: the extension of P with an item y
Output: the utility-list of Pxy
1  UtilityListOfPxy ← ∅;
2  foreach tuple ex ∈ Px.LUList do
3      if ∃ey ∈ Py.LUList and ex.tid = exy.tid then
4          if P.LUList ≠ ∅ then
5              Search element e ∈ P.LUList such that e.tid = ex.tid.;
6              exy ← (ex.tid, ex.iutil + ey.iutil − e.iutil, ey.rutil);
7          else
8              exy ← (ex.tid, ex.iutil + ey.iutil, ey.rutil);
9          end
10         UtilityListOfPxy ← UtilityListOfPxy ∪ {exy};
11     end
12 end
13 return UtilityListPxy;
```

Algorithm 2: The generatePeriods procedure

```
Input: lUl: a LU-list, lMinutil: a user-specified utility threshold, minLength: a user-specified window length threshold
Output: the LU-lists with periods
1  winStart = 0;
2  Find winEnd (the end index of the first window in ul), iutils (sum of iutil values of the first window), rutils (sum of rutils values of the first window);
3  while winEnd < lUl.size do
4      while ul.get(winStart).time is same as previous index do
5          iutils = iutils − lUl.get(winStart).iutil;
6          rutils = rutils − lUl.get(winStart).rutil;
7          winStart = winStart + 1;
8      end
9      while ul.get(winEnd).time ≤ ul.get(winStart).time + minLength do
10         iutils = iutils + lUl.get(winEnd).iutil;
11         rutils = rutils + lUl.get(winEnd).rutil;
12         winEnd = winEnd + 1;
13     end
14     merge the [winStart, winEnd] period with the previous period if iutils ≥ lMinutil. Otherwise, add it to lul.iutilPeriods;
15     merge the [winStart, winEnd] period with the previous period if iutils + rutils ≥ lMinutil. Otherwise, add it to lul.utilPeriods;
16 end
```

The *generatePeriods* procedure (Algorithm 2) takes as input (1) a LU-list lUl, (2) $lMinutil$ and (3) $minLength$. The procedure slides a window over lUl using two variables $winStart$ (initialized to 0; the first element of lUl), and $winEnd$. The procedure first scan lUl to find $winEnd$ (the end index of the first window), $iutils$ (sum of $iutil$ values in the first window) and $rutils$ (sum of $rutil$ values in the first window). Then, it repeats the following steps until the end index $winEnd$ reaches the last tuple of the LU-list: (1) increase the start index $winStart$ until the timestamp changes, and at the same time decrease $iutils$ ($rutils$) by the $iutil$ ($rutil$) values of tuples that exit the current window, (2) increase the end index until the window length is no less than $minLength$, and at same time increase $iutils$ ($rutils$) by the $iutil$ ($rutil$) values of tuples that enter the current window, (3) compare the resulting $iutils$ and $iutils + rutil$ values with

1	2	3	4	5	6	7	8
d_1	d_3	d_3	d_5	d_6	d_7	d_9	d_{10}

Figure 5: The *tid2time* array

lMinutil to determine if the current period should be merged with the previous period or added to *iutilPeriods* and *utilPeriods* (line 14 to 15). Merging is performed to obtain the maximum LHUI and PLHUI periods.

5.3. The LHUI-Miner algorithm

LHUI-Miner takes as input a transaction database with utility values and the *lMinutil* and *minLength* thresholds. The algorithm first scan the database to calculate the TWU of each item. At the same time, an array *tid2time* is constructed, where the *i*-th position stores the timestamp of transaction $t(T_i)$. For example, the *tid2time* array for the example database is shown in Fig. 5. Thereafter, the algorithm only consider items having a TWU no less than *lMinutil*, denoted as I^*. The TWU values of items are used to set a total order $\succ$ on I^*, which is the order of ascending TWU values [12]. A database scan is then performed to reorder items in each transaction according to $\succ$, and build the LU-list of each item $i \in I^*$. Then, the depth-first search of itemsets starts by calling the recursive *LHUI-Search* procedure with $\emptyset$, the LU-lists of 1-itemsets, *lMinutil* and *minLength*.

LHUI-Search (Algorithm 3) takes as input (1) an itemset *P*, (2) a set of extensions of *P*, (3) *lMinutil*, and (4) *minLength*. The procedure then checks if *iutilPeriods* is empty in the LU-list of each extension *Px* of *P*. If yes, *Px* is a LHUI and it is output with its abbreviated maximum LHUI periods (derived from *iutilPeriods* and *tid2time*). Moreover, if *utilPeriods* is not empty, it means that extensions of *Px* should be explored. This is performed by merging *Px* with each extension *Py* of *P* such that $y \succ x$ to form an extension of the form *Pxy* containing $|Px| + 1$ items. The LU-list of *Pxy* is then constructed using the *Construct* procedure of *HUI-Miner*, which join the tuples in the LU-lists of *P*, *Px* and *Py*. Thereafter, *iutilPeriods* and *utilPeriods* in the LU-list of *Pxy* are constructed by calling the *generatePeriods* procedure. Then, *LHUI-Search* is called with *Pxy* to calculate its utility and explore its extension(s) using a depth-first search. The *LHUI-Miner* procedure starts from single items, it recursively explores the search space of itemsets by appending single items and it only prunes the search space based on the properties of LU-list. Hence, it can be easily seen that this procedure is correct and complete to discover all LHUIs.

Algorithm 3: LHUI-Search

```
Input: P: an itemset, ExtensOfP: extensions of P, lMinutil: a user-specified threshold, minLength: a window length threshold
Output: the set of LHUIs and their abbreviated maximum LHUI periods
1  foreach itemset $Px \in ExtensOfP$ do
2      if $Px.LUList.iutilPeriods \neq \emptyset$ then output Px with Px.LUList.iutilPeriods;
3      if $Px.LUList.utilPeriods \neq \emptyset$ then
4          $ExtenssOfPx \leftarrow \emptyset$;
5          foreach itemset $Py \in ExtensOfP$ such that $y \succ x$ do
6              $Pxy.LUList \leftarrow$ Construct (P, Px, Py);
7              generatePeriods (Pxy, lMinutil, minLength);
8              $ExtenssOfPx \leftarrow ExtensOfPx \cup Pxy$;
9          end
10         LHUI-Miner (Px, ExtensOfPx, minutil, minLength);
11     end
12 end
```

5.4. The PHUI-Miner algorithm

PHUI-Miner takes as input a transaction database, *lMinutil*, *minLength* and a positive integer λ, and outputs the PHUIs with their peak windows. The main procedure of PHUI-Miner is the same as LHUI-Miner except that PHUI-Miner has an extra parameter λ and Line 7 of Algorithm 3 is changed to *generatePeak* procedure. This latter procedure maintains two windows v_1 and v_2 instead of one, of length *minLength* and $\lambda \times minLength$, respectively. The *generatePeak* procedure first traverses the LU-List of the current itemset *X* to find $winStart_{v_1}$, $winEnd_{v_1}$ and $winEnd_{v_2}$, $iutils_{v1}$ and $iutils_{v2}$ (sum of *iutil* values for v_1 and v_2), and $rutils_{v1}$ and $rutils_{v2}$ (sum of *rutil* values in v_1 and v_2). Then, the procedure repeats the following steps until v_2 reaches the end index of the LU-list: (1) increase the start indexes $winStart_{v_1}$ and $winStart_{v_2}$ (initialized to 0) until the timestamp changes, and at the same time decrease $iutils_{v_1}$ ($rutils_{v_1}$) and $iutils_{v_2}$ ($rutils_{v_2}$) by the *iutil* (*rutil*) values of tuples that exit the current window, (2) increase the end indexes $winEnd_{v_1}$ and $winEnd_{v_2}$ until the window length is no less than *minLength* or $\lambda \times minLength$, and at the same time increase $iutils_{v_1}$ ($rutils_{v_1}$) and $iutils_{v_2}$ ($rutils_{v_2}$) by the *iutil* (*rutil*) values of tuples that enter the current window, (3) compare $iutils_{v1}$ and $iutils_{v1} + rutil_{v1}$ with *lMinutil* to determine whether to merge the period or add it to *iutilPeriods* and *utilPeriods*, (4) compare $\frac{iutils_{v1}}{lMinutil}$ and $\frac{iutils_{v2}}{\lambda \times lMinutil}$ to determine if the period is a peak window.

Algorithm 4: PHUI-Search

```
Input: $P$ : an itemset, $ExtensOfP$: extensions of $P$, $lMinutil$: a user-specified threshold, $minLength$: a window length threshold, $\lambda$: a user-specified
    parameter
Output: the set of PHUIs and their peak windows
1  for itemset $Px \in ExtensOfP$ do
2      if $Px.LUList.iutilPeriods \neq \emptyset$ then output $Px$ with $Px.LUList.iutilPeriods$;
3      if $Px.LUList.utilPeriods \neq \emptyset$ then
4          $ExtensOfPx \leftarrow \emptyset$;
5          for itemset $Py \in ExtensOfP$ such that $y \succ x$ do
6              $Pxy.LUList \leftarrow$ Construct ($P, Px, Py$);
7              generatePeaks ($Pxy$, $lMinutil$, $minLength$, $\lambda$);
8              $ExtensOfPx \leftarrow ExtensOfPx \cup Pxy$;
9          end
10         PHUI-Miner ($Px$, $ExtensOfPx$, $minutil$, $minLength$);
11     end
12 end
```

Algorithm 5: The generatePeaks procedure

```
input : $lUl$: a LU-list, $lMinutil$: a user-specified utility threshold, $minLength$: a user-specified window length threshold, $\lambda$: a user-specified parameter
output: the LU-lists with periods
1  $winStart_{v1} = 0$, $winStart_{v2} = 0$;
2  Find $winEnd_{v1}$ $winEnd_{v2}$ (the end index of first v1 and v2 in $ul$), $iutils_{v1}$ and $iutils_{v2}$ (sum of $iutil$ values of the first v1 and v2), $rutils_{v1}$ and $rutils_{v2}$ (sum of
    $rutils$ values of the first v1 and v2);
3  while $winEnd < lUl.size$ do
4      while $ul.get(winStart).time$ is same as previous index do
5          $iutils = iutils - lUl.get(winStart).iutil$;
6          $rutils = rutils - lUl.get(winStart).rutil$;
7          $winStart_{v1} = winStart_{v1} + 1$;
8          $winStart_{v2} = winStart_{v2} + 1$;
9      end
10     while $ul.get(winEnd).time \leq ul.get(winStart).time + minLength$ do
11         $iutils = iutils + lUl.get(winEnd).iutil$;
12         $rutils = rutils + lUl.get(winEnd).rutil$;
13         $winEnd_{v1} = winEnd_{v1} + 1$;
14     end
15     while $ul.get(winEnd).time \leq ul.get(winStart).time + minLength \times \lambda$ do
16         $iutils = iutils + lUl.get(winEnd).iutil$;
17         $rutils = rutils + lUl.get(winEnd).rutil$;
18         $winEnd_{v2} = winEnd_{v2} + 1$;
19     end
20     merge the $[winStart, winEnd]$ period with the previous period if $iutils \geq lMinutil$. Otherwise, add it to $lul.iutilPeriods$;
21     merge the $[winStart, winEnd]$ period with the previous period if $iutils + rutils \geq lMinutil$. Otherwise, add it to $lul.utilPeriods$;
22     compare $\frac{iutils_{v1}}{lMinutil}$ and $\frac{iutils_{v2}}{\lambda \times lMinutil}$ to determine if the period is a peak window;
23 end
```

5.5. The NPHUI-Miner algorithm

The NPHUI-Miner algorithm (Algorithm 6) takes as input a transaction database, *lMinutil*, *minLength* and the moving average crossover coefficient λ, and outputs the NPHUIs with their peak windows. The main procedure of NPHUI-Miner is the same as PHUI-Miner except that a post-processing step is applied to eliminate PHUIs that are not NPHUIs. This is done by calling the *FindNPHUIs* procedure with the set of PHUIs. This procedure first sort the PHUIs by length (line 1). Then, it checks if each itemset containing more than one item is non redundant (line 2 to 13). The algorithm does not need to check if itemsets containing one item are redundant because their peak windows are by definition non redundant peak windows. To check if other PHUIs are non redundant, the algorithm considers each

PHUI by order of ascending length. For an itemset p of length i, a loop is performed to compare its peak windows with each subset $q \subset p$ of length $i - 1$ (line 4 to 10). If q is not a PHUI, then p is redundant and is removed from the set of PHUIs (line 9). Otherwise, if q is a PHUI, then the peak windows of p and q are compared (line 5 to 8). Each peak window of p that is not overlapping with a peak window of q is removed from the set of peak windows of p because it is a redundant peak window (line 6). After such removal, if p has no peak windows left, then p is removed from the set of PHUIs (line 7). After performing these steps, the set of PHUIs contains the set of NPHUIs, which is returned to the user.

To improve the performance ot the *FindNPHUIs* procedure, a stopping criterion has been added on line 12. It is based on Theorem 5, which states that if there are less than $i + 1$ PHUIs of any length i, then all PHUIs of length greater than i are not NPHUIs. If that criterion is satisfied, all PHUIs containing more than i items are eliminated.

Algorithm 6: The FindNPHUIs algorithm

```
Input: PHUIs: the peak high-utility itemsets with their peak windows
Output: the non redundant peak high-utility itemsets with their non redundant peak windows
1  Sort PHUIs by length;
2  for $i = 2$ to the length of the largest PHUI do
3      for each PHUI $p$ of length $i$ do
4          for each itemset $q \subset p$ of length $i - 1$ do
5              if $q$ is a PHUI then
6                  Remove each window of $p$ that is not overlapping with a peak window of $q$;
7                  if $p$ has no peak windows left then Remove $p$ from the set of PHUIs;
8              else
9                  Remove $p$ from the set of PHUIs;
10             end
11         end
12     end
13     if there is less than $i + 1$ PHUIs of length $i$ then remove all itemsets having a length greater than $i$ from the set PHUIs;
14 end
15 Return PHUIs;
```

5.6. Optimizations

To improve the performance of LHUI-Miner and PHUI-Miner, the next paragraphs describe three optimizations.

Strategy 1 (Discarding unpromising items using the sliding window). During the first database scan, if there is an item i such that for any window $W_{k,l}$ of length *minLength*, $TWU_{k,l}(i) < lMinUtil$, then item i is discarded.

Theorem 9. A transaction T is said to be irrelevant if for each item $i \in T$ and window $W_{k,l} \supset T$ such that $length(W_{k,l}) = minLength$, $TWU_{k,l}(i) < lMinutil$. For any LHUI X and transaction T included in a LHUI period of X, $u(X, T) = 0$, i.e. T does not contribute to the utility of X in that LHUI period.

Proof 13. A proof by contradiction is made. Consider that T is in a LHUI period $W_{k,l}$ ($length(W_{k,l}) = minLength$) of itemset X and $u(X, T) > 0$. Since $u(X, T) > 0$, $X \subseteq T$. Moreover, $\forall i \in X, TWU_{k,l}(i) \geq u_{k,l}(X) \geq lMinutil$ by Lemma 1. Because $X \subseteq T$, $i \in T$ and there is a contradiction with $\forall i \in T$, $TWU_{k,l}(i) < lMinutil$. Thus, the theorem holds. □

Strategy 2 (Discarding irrelevant transactions). During the first database scan, all irrelevant transactions are identified. Then, they are ignored when constructing the LU-lists of itemsets because an irrelevant transaction cannot contribute to the utility of any LHUI in its LHUI periods (by Theorem 9).

Theorem 10. A transaction T that is not in any PLHUI period of an itemset X is called an irrelevant transaction w.r.t X and its transitive extensions. It follows that transaction T is not in the LHUI periods of any transitive extension Y of X.

Proof 14. By Theorem 7, if any window $W_{k,l}$ of length *minLength* containing the transaction T is not a PLHUI period of itemset X, then $W_{k,l}$ is not a LHUI period for any transitive extension Y of X. Thus, transaction T cannot be in any LHUI period of Y. □

Strategy 3 (Discarding unpromising tuples in each LU-list). The LU-list of an itemset X can store numerous tuples that represents the transactions where X appears. This strategy consists of not storing the tuples corresponding to irrelevant transactions w.r.t. X and its transitive extensions (based on Theorem 10). This reduces the runtime of the algorithms since performing the intersection of LU-lists and scanning LU-lists is faster for smaller LU-lists. This strategy is applied during LU-list construction.

5.7. Complexity

The complexity of the proposed algorithms can be analyzed as follows. First, consider the LHUI-Miner algorithm. It first scans the database twice to calculate the TWU of items, create the *tid2time* array, and build the LU-lists of items. The time cost of each database scan is roughly $O(w)$ time where w is the number of transactions in the database. Then, the algorithm performs a recursive exploration of the search space by calling the *LHUI-search* procedure for each itemset that is considered in the search space.

For each itemset having more than one item, the time cost of constructing the LU-list of an itemset Pxy is the time required for applying the *Construct* procedure and that of applying the *generatePeriods* procedure. The former procedure requires $O(m + n + o)$ time if implemented using a three-way comparison, where m, n and o are the number of tuples in the LU-lists of Px, Py and P, respectively (see [25] for details on this optimization). The latter procedure requires to scan a LU-list once to calculate the periods of an itemset Pxy. Thus, the time complexity of *generatePeriods* is $O(q)$ where q is the number of tuples in the LU-list of Pxy. Hence, the time cost of processing an itemset Pxy is roughly $O(m + n + o + q)$.

The number of itemsets considered by LHUI-Miner depends on how the user sets the algorithm's parameters, and how effective the pruning strategies are at reducing the search space given these parameter values. In the worst case, where no itemsets can be pruned from the search space, the algorithm will consider all the $2^{|I|} - 1$ possible itemsets. However, in practice the search space can be considerably reduced thanks to the pruning strategies.

In terms of space cost, the algorithm requires to store the *tid2time* array which requires $O(w)$ space (one entry for each transaction). Moreover, a LU-list is created for each considered itemset in the search space. In the worst case, a LU-list contains a tuple and period information for each transaction, and thus requires $O(w)$ space. But generally, LU-lists can be quite small as items often do not appear in all transactions.

Thus, on overall, the time and space complexity required by LHUI-Miner to process each itemset is linear, and the number of itemsets depends on how parameters are set. The PHUI-Miner algorithm performs similar steps to LHUI-Miner and thus has similar complexity.

The NPHUI-Miner algorithm performs a post-processing step after applying PHUI-Miner by calling the $FindNPHUIs$ procedure. The complexity of that procedure is in the worst case as follows. Assumes that there are n single items. In the worst case, all the supersets of single items are PHUIs, that is there are C_n^k PHUIs containing k items ($1 \leq k \leq n$). For each k-itemset ($k > 1$), the procedure checks if its k (k-1)-subsets are NPHUIs. Thus, totally, for all itemsets from size 1 to n, the number of checks is $n + (2 \times C_n^2) + (3 \times C_n^3) + \cdots + (n \times C_n^n) = n + 2 \times \frac{P_n^2}{2!} + 3 \times \frac{P_n^3}{3!} + \cdots + n \times \frac{P_n^n}{n!} = n(1 + \frac{(n-1)}{1!} + \frac{(n-1)(n-2)}{2!} + \cdots + \frac{(n-1)!}{(n-1)!} = n(1 + C_{n-1}^1 + C_{n-1}^2 + \cdots + C_{n-1}^{n-1}) = n \times 2^{(n-1)}$. Thus, the complexity of the post-processing step performed by NPHUI-Miner is in the worst case $O(n \times 2^n)$. However, as it will be shown in the experiments, the cost of that step is small compared to the extraction of PHUIs, and as a result, the execution time of NPHUI-Miner is similar to that of PHUI-Miner.

6. Experimental Evaluation

Experiments were performed to assess the performance of LHUI-Miner, PHUI-Miner and NPHUI-Miner on a computer having an Intel Xeon E3-1270 v5 processor running Windows 10, and 16 GB of free RAM. Since the proposed algorithms are designed for new pattern mining problems, there is no algorithm that can be directly compared with them. For this reason, the performance of LHUI-Miner and PHUI-Miner were compared with non optimized versions of these algorithms by deactivating optimizations. Moreover, the number of LHUIs, PHUIs and NPHUIs found was compared with the number of high utility itemsets found by the traditional HUI-Miner algorithm, which discovers all high utility itemsets. Note that the performance of HUI-Miner is not directly compared with the proposed algorithms because it addresses a different problem, which is easier than the problem of mining LHUIs, PHUIs and NPHUIs.

Four real-life datasets commonly used in the HUIM litterature were used: *mushroom*, *retail*, *kosarak* and *e-commerce*, obtained from `http://www.philippe-fournier-viger.com/spmf/index.php?link=datasets.php`. They represent the main types of data typically encountered in real-life scenarios (dense, sparse, and long transactions). Let $|I|$, $|D|$ and A represents the number of distinct items, transactions and average transaction length. *mushroom* is a dense dataset ($|I|$ = 16,470, $|D|$ = 88,162, A = 23). *kosarak* is a dataset that contains many long transactions ($|I|$ = 41,270, $|D|$ = 990,000, A = 8.09). *retail* is a sparse dataset with many different items ($|I|$ = 16,470, $|D|$ = 88,162, A = 10,30). *e-commerce* is a real-world dataset ($|I|$ = 3,803, $|D|$ = 17,535, A = 15.4), containing customer transactions from 01/12/2010 to 09/12/2011 of an online store. Note that transactions containing more than 100 items were deleted from that dataset since they are large orders made by companies rather than individual customers. Also, items with a utility greater than 1,000 times the average were removed. For the other datasets, the external utility of items are generated between 1 and 1,000 using a log-normal distribution and quantities of items are generated randomly between 1 and 5, as in [25, 35]. Besides, the timestamps of transactions in these three databases are generated by adopting the same distribution as the real e-commerce database. The source code of algorithms and datasets can be downloaded from `http://www.philippe-fournier-viger.com/spmf/`. Memory measurements were done using the standard Java API.

In terms of parameter settings, LHUI-Miner, PHUI-Miner and NPHUI-Miner were run with $minLength$ = 90 *days* for *e-commerce* and 30 *days* for the other datasets. For PHUI-Miner and NPHUI-Miner, $\lambda = 2$ was used. Thereafter, *lhui-op* denotes LHUI-Miner with optimizations; *lhui-non-op* denotes LHUI-Miner without optimization; *phui-op* denotes PHUI-Miner with optimizations; and *nphui-op* denotes NPHUI-Miner with optimizations.

6.1. Influence of the lMinutil parameter

The first experiment evaluates the influence of $lMinutil$ on the runtime and number of patterns. Algorithms were run on each dataset, while decreasing $lMinutil$ until they became too long to execute, ran out of memory or a clear trend was observed. The HUI-Miner was run with $minutil = lMinutil \times \lceil \frac{W_D}{minLength} \rceil$ to find high utility itemsets. Fig. 6 compares the execution times of PHUI-Miner and LHUI-Miner with and without optimization. Fig. 7 compares the numbers of LHUIs, PHUIs and HUIs found by the algorithms.

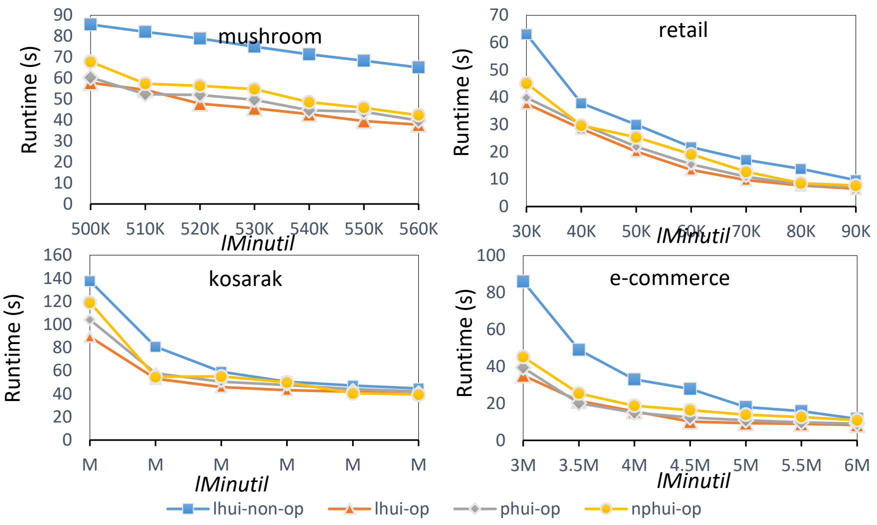

Figure 6: Execution times for different $lMinutil$ values

It can be observed that in most cases, optimizations reduce the runtime. In some cases, optimized algorithms are one time faster than the non-optimized algorithm, while in some cases the improvement is smaller. The execution time of PHUI-Miner can be a little bit longer than LHUI-Miner for the same parameters, and NPHUI-Miner is a little bit slower than PHUI-Miner since it removes redundant PHUIs by postprocessing.

A second observation is that the number of LHUIs, PHUIs and NPHUIs is much more than the number of HUIs in most cases. This is reasonable since an itemset is much more likely to be high utility in at least one window

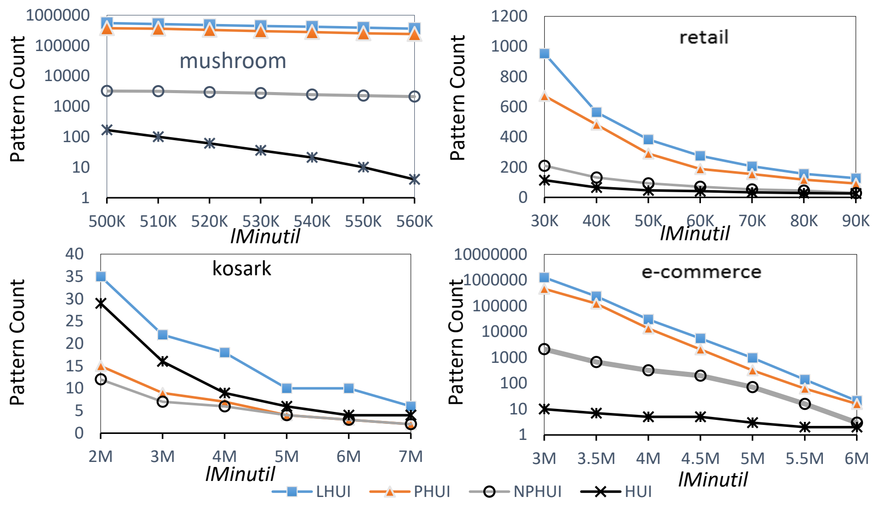

Figure 7: Number of patterns found for different *lMinutil* values

	mushroom	*retail*	*kosarak*	*e-commerce*
lhui-no-op	58-104	173-427	6-35	121-218
lhui-op	32-98	156-412	2-15	103-176
phui-op	28-101	163-400	3-13	98-183
nphui-op	29-103	165-401	3-14	99-185

Table 5: Memory consumption range (MB)

than in the whole database. For example, on mushroom (W_D = 180 *days*), *minutil* = 500,000, *lMinutil* = 83,333, *minlength* = 30 *days*, there are 168 HUIs, 549,479 LHUIs, 372,583 PHUIs and 3,209 NPHUIs. It also can be seen that the number of PHUIs is always less than the number of LHUIs. The main reason is that the set of peak windows of an itemset are a subset of its LHUI periods. And the number of NPHUIs is fewer than that of PHUIs since NPHUI-Miner eliminates many redundant PHUIs.

6.2. *Influence of the minlength parameter*

The second experiment evaluates the influence of *minlength* on runtime and number of patterns. To assess the influence of *minLength*, the parameter was varied from 7 *days* to 360 *days*, and the execution time and number of patterns was measured for LHUI-Miner with or without optimizations. In this experiment, *lMinutil* is respectively set to 500,000, 30,000, 2,000,000 and 3,000,000 for *mushroom*, *retail*, *kosarak* and *e-commerce*, and *minLength* = 360. For other *minLength* values, we decreased the *lMinutil* threshold to preserve the same average utility. Because the runtime of PHUI-Miner and NPHUI-Miner are very similar to that of LHUI-Miner, only the performance of LHUI-Miner is compared. Fig. 8 and Fig. 9 show the execution time and number of pattern for different *minLength* values, respectively.

It can be observed that the execution time and number of patterns decrease when the *minLength* parameter is increased. It is reasonable since using larger windows means that utility changes are more smoothed. Another observation is that as the *minLength* threshold is increased, the time and number of patterns decreases more and more slowly. This is because when the *minLength* parameter is set to large values, the average utility in a window is not much different from the average utility of the whole database.

6.3. *Influence of the lambda parameter*

The third experiment evaluates the influence of the λ parameter on runtime and number of patterns. The λ parameter was varied from 2 to 10 to evaluate how it influences the number of PHUIs and NPHUIs. In this experiment, *minLength* = 30 *days*, and *lMinutil* is set to 45,000, 2,500, 180,000 and 250,000 for *mushroom*, *retail*, *kosarak*

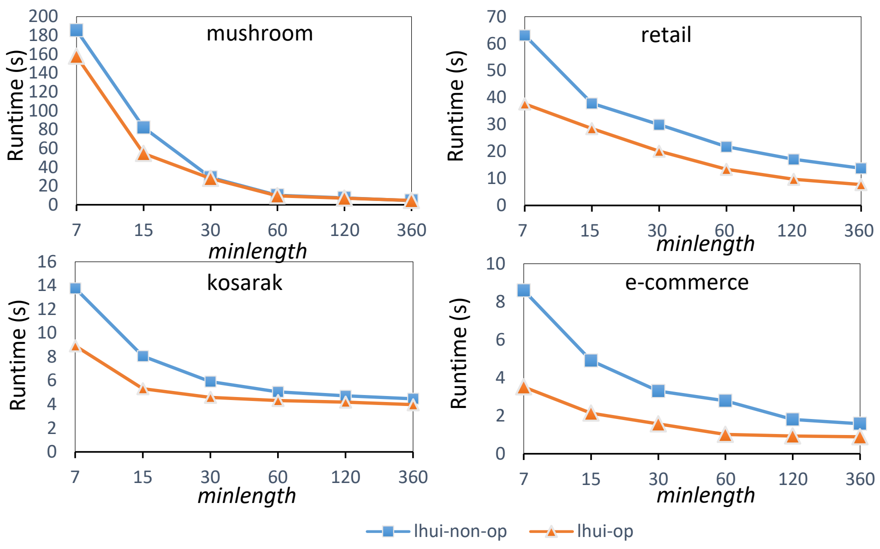

Figure 8: Execution times for different *minLength* values

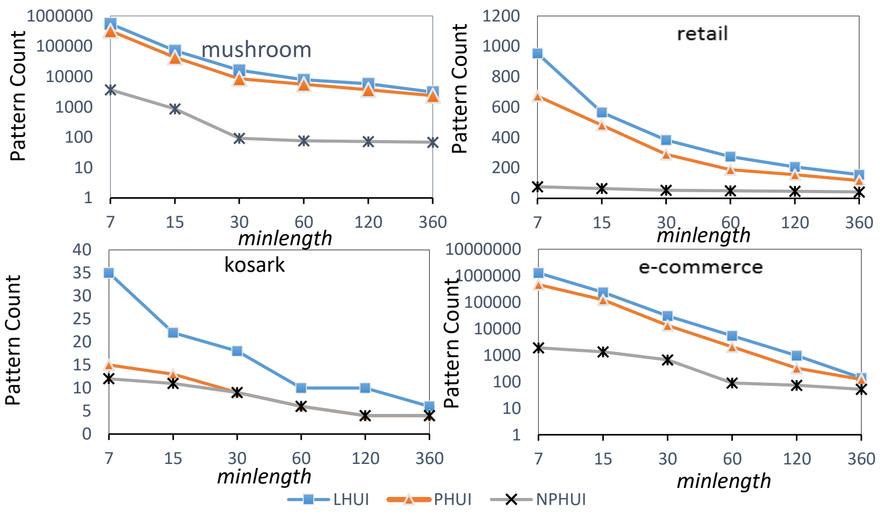

Figure 9: Patterns found for different *minLength* values

and *e-commerce*, respectively. Fig. 10 shows the number of patterns found by PHUI-Miner and NPHUI-Miner. The execution times are not shown because they are the same. It can be observed that the number of patterns increases when λ is increased. It is because when the λ parameter is set to small values, the short and long term moving averages are less smooth and tend to cross each other more often. Thus, there are many small periods that have a length less than *minLength*, which are not PHUI period. And when $\lambda = \infty$, PHUIs are LHUIs.

6.4. Comparison of memory consumption

Memory consumption was also measured. The maximum memory usage for each dataset is shown in Table 5 for each algorithm as a range for all its executions. The memory consumption of NPHUI-Miner is almost the same as PHUI-Miner as it only performs an additional post-processing step. It is found that optimized algorithms generally use less memory than non-optimized algorithms. For example, for *mushroom* with $lMinutil = 83,333$ and $minlength = 30$ *days*, lhui-non-op, lhui-op and phui-op consume 58 MB, 32 MB and 28 MB, respectively. The main reason for that reduced memory usage is that Strategy 3 remove elements from LU-lists. Hence, less information is kept in memory.

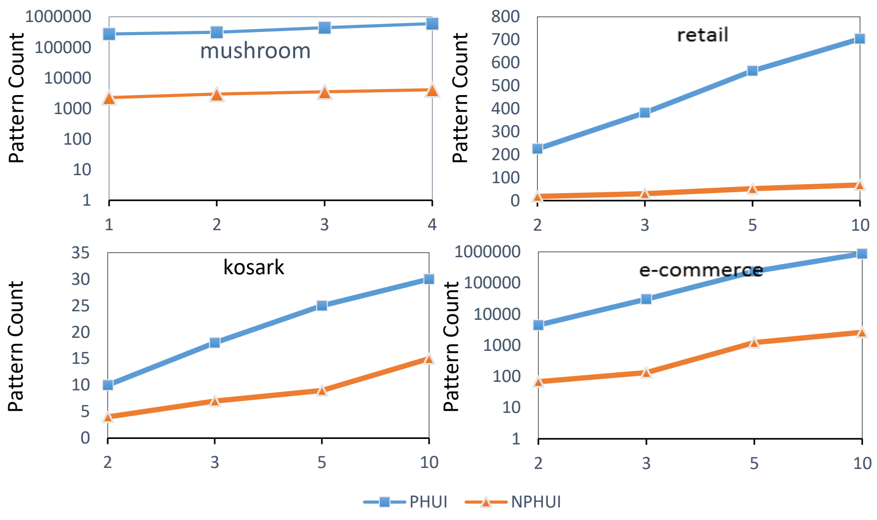

Figure 10: Patterns found for different λ values

Furthermore, LHUI-Miner and PHUI-Miner have similar memory consumption because they scan the same candidate LU-lists.

Overall, the performance of the proposed algorithms can be considered as satisfactory to analyze datasets of up to 1,000,000 transactions on a desktop computer, and optimizations were shown to improve the performance.

6.5. *Runtime comparison with HUI-Miner*

The runtime of the proposed algorithms was also compared with that of HUI-Miner for $lMinutil \times \lceil \frac{W_D}{minLength} \rceil$. This comparison was done because the proposed algorithm extend the basic search procedure of HUI-Miner.

It was observed that in some cases the proposed algorithms can be much slower than HUI-Miner. The reason is that HUI-Miner generally explores a smaller search space and generates much fewer patterns compared to the proposed algorithms (as shown in Fig. 7 and discussed in Section 6.1). This is in accordance with the theorems presented in this paper about the relationships between the traditional problem of HUIM and the proposed problems. The number of NPHUI is often similar to that of HUI. But NPHUIs provides information about peaks.

6.6. *Analysis of some discovered patterns*

To further illustrate the usefulness of the proposed pattern definitions, this subsection discusses some patterns found in the *e-commerce* dataset, containing data from $W_D = 375$ days of sales. Consider that $lMinutil = 665,000$, $minLength = 50\ days$. It is found that the itemsets $\{retro\_spot\_bag\}$ and $\{retro\_spot\_bag, polka\_dot\_bag\}$ are two LHUIs. If we apply a traditional HUI mining algorithm with $minutil = lMinutil \times \lceil \frac{W_D}{minLength} \rceil = 665,000 \times \lceil 375/50 \rceil = 5,320,000$, the itemset $\{retro\_spot\_bag, polka\_dot\_bag\}$ is not a high utility itemset. Hence, this itemset is not presented to the user because it does not have a high utility in the whole database, although it has a high utility in some specific time periods, which is useful information for market basket analysis.

To better understand these results, Figure 11 depicts the utility distribution of the itemsets $\{polka\_dot\_bag\}$, $\{retro\_spot\_bag\}$, and $\{retro\_spotbag, polka\_dot\_bag\}$ from day 1 to day 281. For each itemset, the utility is shown using moving averages for windows of length 50 and 90 respectively (mau_{50} and mau_{90}). On that figure, a black line represents the average utility for the database, which is calculated as $minutil/W_D = 5,320,000/375 = 14,187$.

The itemset $\{retro\_spot\_bag, polka\_dot\_bag\}$ has a LHUI period from day 169 to day 281. This can be seen in Figure 11 as the utility of that itemset is above the average (black line) from approxmiately day 169 to day 281. The itemset $\{retro\_spot\_bag, polka\_dot\_bag\}$ is not a high utility itemset since the utility of that itemset is generally below the black line. The itemset $\{polka\_dot\_bag\}$ is not a high utility itemset and LHUI since the utility of that itemset is below the black line for all but a few days. A high utility itemset is expected to have an area under the curve that is greater than the area under the black line.

By applying the PHUI-Miner algorithm with the same parameters and $\lambda = 1.8$, it is found that the itemset $\{retro\_spot\_bag\}$ has a peak window from day 180 to day 257. The begining and end of this peak window occur when the short and long term moving average (mau_{50} and mau_{90}) cross, as indicated by the black arrows on Figure 11. This information indicates that the itemset $\{retro\_spot\_bag\}$ yields higher profit than usual during that period. This can be useful for the manager of a store, for example to refill stocks of the item $\{retro\_spot\_bag\}$ before the 180^{th} *day* of the year to prepare for higher sales, and to make some promotion after the 257^{th} *day* to reduce stocks given that sales are expected to decrease.

Similarly, the itemset $\{retro\_spot\_bag, polka\_dot\_bag\}$ has a peak window from day 171 to day 227. While the itemset $\{retro\_spot\_bag\}$ is a NPHUI, the itemset $\{retro\_spot\_bag, polka\_dot\_bag\}$ is not a NPHUI since its subset $\{polka\_dot\_bag\}$ is not a PHUI. This means that the itemset $\{polka\_dot\_bag\}$ does not considerably contribute to the peak observed for the itemset $\{retro\_spot\_bag, polka\_dot\_bag\}$.

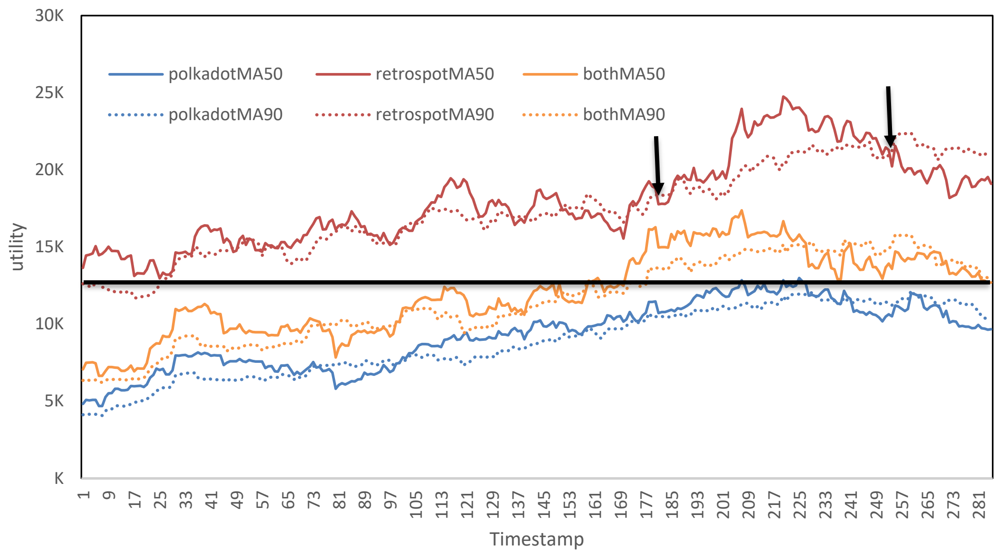

Figure 11: Utility distribution of three itemsets for the *e-commerce* database from day 1 to day 281

On overall, it was observed that the proposed algorithms can discover useful patterns that provide information about the peaks of utility that cannot be found by traditional high utility itemset mining algorithms. In market basket analysis, such patterns can be used to understand the behavior of customers, but the developped algorithms can also be applied in other domains where data is modelled as sets of items (transactions) with weights (utility).

Besides using patterns to understand the data, paterns could also be used in other ways. For example, high utility patterns could be used for building a product recommendation system considering the utility of patterns (profit) to recommend products to customers, or to build a forecasting system to predict future peaks. In that context other aspects could be evaluated such as the relevance and accuracy of recommendations/predictions generated using the patterns. Another application of extracting patterns could be to build features to train classification models or to cluster patterns having similar peaks. These ideas could be studied in future work.

7. Conclusion

To find itemsets that yield a high utility in non-predefined time periods and consider timestamps of transactions, this paper defined the problem of mining Local High-Utility Itemsets (LHUIs), and extended it to find Peak High-Utility Itemsets (PHUIs), to find the time periods where an itemset generates a utility that is considerably higher than usual. Moreover, a third problem was proposed to reduce redundancy in the set of PHUIs, which is to mine non-redundant peak high utility itemsets. Three algorithms named LHUI-Miner (Local High-Utility Itemset Miner), PHUI-Miner (Peak High-Utility Itemset Miner) and NPHUI-Miner (Non redundant Peak High-Utility Itemset Miner) were designed to efficiently discover LHUIs, PHUIs and NPHUIs. Besides, three strategies were proposed to improve the performance of these algorithms. An experimental evaluation has shown that the algorithms can discover useful patterns that traditional HUIM could not find and that strategies reduces the runtime and memory consumption.

For future work, several research directions can be explored. A possibility is to consider user preferences over time windows. This can be done by adding weights to transactions of preferred periods. These weights could then be considered when computing the utility of patterns. Other types of preferences could also be considered. Another possibility is to adapt the concept of peak to other pattern mining problems such as episode mining and sequential pattern mining. It would aso be possible to adapt the concept of peak to stream mining, and design a single phase algorithm that could find the NPHUIs directly without doing post-processing. Besides, the proposed algorithms could be parallelized to decrease runtime. Since the algorithms performs a depth-first search, each branch of the search space can be explored independently. Lastly, it would also be interesting to define a method to automatically set parameters such as the window length. But dynamically adjusting this latter parameter is challenging as it would require to define novel upper-bounds that are valid for various window lengths while not being too loose to still be able to reduce the search space and mine patterns efficiently.

References

[1] Agrawal, R., Srikant, R.: Fast algorithms for mining association rules in large databases. In: Proc. 20th Int. Conf. Very Large Databases, pp. 487–499, Morgan Kaufmann, Santiago de Chile (1994)

[2] Aggarwal, C. C. Data Mining The Textbook, Springer (2015)

[3] Ahmed, C. F., Tanbeer, S. K., Jeong, B.: Mining High Utility Web Access Sequences in Dynamic Web Log Data, In: Proc. of 11th ACIS International Conference on Software Engineering, Artificial Intelligences, Networking and Parallel/Distributed Computing, pp.76–81. IEEE, London (2010)

[4] Alkan, O. K., Karagoz, P.: Crom and huspext: Improving efficiency of high utility sequential pattern extraction. IEEE Trans. on Knowledge and Data Engineering, 27(10), 2645–2657, (2015)

[5] Barsky, M., Kim, S., Weninger, T., Han, J.: Mining flipping correlations from large datasets with taxonomies. VLDB Endowment, 5(4), 370–381 (2011)

[6] Duong, Q. H., Fournier-Viger, P., Ramampiaro, H., Norvag, K. Dam, T.-L.: Efficient High Utility Itemset Mining using Buffered Utility-Lists. Applied Intelligence, 48(7), 1859–1877, Springer, (2018)

[7] Farzanyar, Z., Kangavari, M., Cercone, N.: Max-FISM: Mining (recently) maximal frequent itemsets over data streams using the sliding window model. Computers and Mathematics with Applications, 64(6), 1706–1718 (2012)

[8] Fournier-Viger, P., Li, X., Yao, J., Lin, J. C.-W.: Interactive Discovery of Statistically Significant Itemsets. In: Proc. 31rd Intern. Conf. on Industrial, Engineering and Other Applications of Applied Intelligent Systems (IEA AIE 2018), Springer LNAI, pp. 101–113 (2018)

[9] Fournier-Viger, P., Lin, J. C.-W. , Dinh, T., Le, H. B.: Mining Correlated High-Utility Itemsets using the Bond Measure. In: Proc. Intern. Conf. Hybrid Artificial Intelligence Systems. pp. 53–65, Springer Seville, (2016)

[10] Fournier-Viger, P., Lin, J. C.-W., Kiran, R. U., Koh, Y. S., Thomas, R.: A Survey of Sequential Pattern Mining. Data Science and Pattern Recognition (DSPR), vol. 1(1), 54–77 (2017)

[11] Fournier-Viger, P., Lin, J. C.-W., Vo, B, Chi, T.T., Zhang, J., Le, H. B.:A Survey of Itemset Mining. WIREs Data Mining and Knowledge Discovery, e1207 doi: 10.1002/widm.1207 (2017)

[12] 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 Int. Symp. on Methodologies for Intell. Syst., pp. 83–92 Springer, Roskilde (2014)

[13] Fournier-Viger, P., Zida, S.: FOSHU: faster on-shelf high utility itemset mining–with or without negative unit profit. In: Proc. 30th Annual ACM Symposium on Applied Computing, pp. 857–864 ACM, Salamanca (2015)

[14] Gan, W., Lin, J. C. W., Fournier-Viger, P., Chao, H. C.: Mining Recent High-Utility Patterns from Temporal Databases with Time-Sensitive Constraint. In: Int. Conf. on Big Data Analytics and Knowledge Discovery. pp. 3–18, Springer, Porto (2016)

[15] Geng, L., Hamilton, H. J.: Interestingness measures for data mining: A survey. ACM Computing Surveys. 38(3), 61-93 (2006)

[16] Guo, G., Zhang, L., Liu, Q., Chen, E., Zhu, F., Guan, C.: High utility episode mining made practical and fast. In: Int. Conf. on Advanced Data Mining and Applications, pp. 71–84, Springer, Cham (2014).

[17] Han, J., Pei, J., Yin, Y., Mao, R.: Mining frequent patterns without candidate generation: A frequent-pattern tree approach. Data mining and knowledge discovery, 8(1), 53–87 (2004)

[18] Kim, D., Yun, U.: Mining high utility itemsets based on the time decaying model. Intelligent Data Analysis, 20(5), 1157–1180 (2016)

[19] Krishnamoorthy, S.: Pruning strategies for mining high utility itemsets. Expert Systems with Applications, 42(5), 2371–2381 (2015)

[20] Li, Y., Kubat, M.: Searching for high-support itemsets in itemset trees. Intelligent Data Analysis. 10(2), 105–120 (2006)

[21] Lin, J. C. W., Gan, W., Hong, T. P., Tseng, V. S.: Efficient algorithms for mining up-to-date high-utility patterns. Advanced Engineering Informatics, 29(3), 648–661 (2015)

[22] Lin, Y. F., Wu, C. W., Huang, C. F., Tseng, V. S.: Discovering utility-based episode rules in complex event sequences. Expert Systems with Applications, 42(12), 5303–5314, (2015)

[23] Lin, J. C. W., Zhang, J., Fournier-Viger, P., Hong, T.P., Zhang, J.: A two-phase approach to mine short-period high-utility itemsets in transactional databases. Advanced Engineering Informatics, 33, 29–43 (2017)

[24] Liu, Y., Liao, W., Choudhary, A.: A two-phase algorithm for fast discovery of high utility itemsets. In: Proc. 9th Pacific-Asia Conf. on Knowl. Discovery and Data Mining, pp. 689–695 Springer, Hanoi (2005)

[25] Liu, M., Qu, J.: Mining high utility itemsets without candidate generation. In: Proc. 22nd ACM Int. Conf. Info. and Know. Management, ACM, pp. 55–64 (2012)

[26] Liu, J., Wang, K., Fung, B.: Direct discovery of high utility itemsets without candidate generation. Proc. 12th IEEE Intern. Conf. Data Mining, pp. 984–989 (2012)

[27] Mannila, H., Toivonen, H., Verkamo, A.I.: Discovery of frequent episodes in event sequences. Data mining and knowledge discovery. 1(3), 259-289 (1997)

[28] Ni, Y., Liao, Y.C., Huang, P.: MA trading rules, herding behaviors, and stock market overreaction. Int. Review of Economics & Finance, 39, 253–265 (2015)

[29] Omiecinski, E.: Alternative Interest Measures for Mining Associations in Databases. IEEE Transactions on Knowledge Discovery and Data Engineering. 15(1), 57–69 (2003)

[30] Pei, J., Han, J., Lu, H., Nishio, S., Tang, S., Yang, D.: H-mine: Hyper-structure mining of frequent patterns in large databases. In: Proc. 2001 IEEE Intern. Conf. Data Mining, pp 441–448, IEEE Computer Society, San Jose (2001)

[31] Peng, A. Y., Koh, Y. S., Riddle, P.: mHUIMiner: A Fast High Utility Itemset Mining Algorithm for Sparse Datasets. In: Proc. 22nd Pacific-Asia Conf. on Knowl. Discovery and Data Mining, pp. 196–207 ACM, Jeju (2017)

[32] Shin, S. J., Lee, D. S., Lee, W. S.: CP-tree: An adaptive synopsis structure for compressing frequent itemsets over online data streams, Information Sciences, 278, 559–576 (2014)

[33] Soulet, A., Raissi, C., Plantevit, M., Cremilleux, B.: Mining dominant patterns in the sky. In: Proc. 11th IEEE Int. Conf. on Data Mining, pp. 655–664, IEEE Computer Society, Vancouver, (2011)

[34] Tang, L., Zhang, L., Luo, P., Wang, M.: Incorporating occupancy into frequent pattern mining for high quality pattern recommendation. In: Proc. 21st ACM Intern. Conf. Information and knowledge management, pp. 75–84 ACM, Maui (2012)

[35] Tseng, V. S., Shie, B.-E., Wu, C.-W., Yu., P. S.: Efficient algorithms for mining high utility itemsets from transactional databases. IEEE Trans. Knowl. Data Eng. 25(8), 1772–1786 (2013)

[36] Uno, T., Kiyomi, M., Arimura, H.: LCM ver. 2: Efficient mining algorithms for frequent/closed/maximal itemsets. In: Proc. ICDM'04 Workshop on Frequent Itemset Mining Implementations, Vol. 126, CEUR, Brighton (2004)

[37] Wan, Q., An, A.: Discovering Transitional Patterns and Their Significant Milestones in Transaction Databases. IEEE Trans. Knowl. Data Eng. 21(12), 1692–1707 (2009)

[38] Wu, C. W., Lin, Y. F., Yu, P. S., Tseng, V. S.: Mining high utility episodes in complex event sequences. In: Proc. of the 19th ACM SIGKDD Int. conf. on Knowledge discovery and data mining, pp. 536–544, ACM (2013)

[39] Xiong, H., Tan, P. N, Kumar, V.: Mining strong affinity association patterns in data sets with skewed support distribution. In: Proc. 2003 IEEE Intern. Conf. Data Mining. pp. 387–394, IEEE Computer Society (2003)

[40] Yin, J., Zheng, Z., Cao, L.: USpan: an efficient algorithm for mining high utility sequential patterns. In Proc. of the 18th ACM SIGKDD Int. conf. on Knowledge discovery and data mining, pp. 660–668, ACM (2012)

[41] Yun, U., Kim, D., Yoon, E., Fujita, H.: Damped Window based High Average Utility Pattern Mining over data streams. Knowledge-Based Systems, 144(15), 188–205 (2018)

[42] Zaki, M. J.: Scalable algorithms for association mining. IEEE Trans. Knowl. Data Eng. 12(3), 372–390 (2000)

[43] Zida, S., Fournier-Viger, P., Lin, J. C.-W., Wu, C.-W., Tseng, V.S.: EFIM: A Highly Efficient Algorithm for High-Utility Itemset Mining. In: Proc. 14th Mexican Int. Conf. on Artificial Intelligence, pp. 530–546, Springer (2015)

[44] Zida, S., Fournier-Viger, P., Wu, C. W., Lin, J. C. W., Tseng, V. S.: Efficient mining of high-utility sequential rules. In: Int. Workshop on Machine Learning and Data Mining in Pattern Recognition, pp. 157–171, Springer (2015)

[45] Zimmermann, A.: Understanding episode mining techniques: Benchmarking on diverse, realistic, artificial data. Intelligent Data Analysis. 18(5), 761–791 (2014)