Mining Locally Trending High Utility Itemsets

Philippe Fournier-Viger[1], Yanjun Yang[2],
Jerry Chun-Wei Lin[3], and Jaroslav Frnda[4]

[1] School of Natural Sciences and Humanities,
Harbin Institute of Technology (Shenzhen), Shenzhen, China
[2] School of Computer Sciences and Technology,
Harbin Institute of Technology (Shenzhen), Shenzhen, China
[3] Department of Computing, Mathematics and Physics, Western Norway University of Applied Sciences (HVL), Bergen, Norway
[4] Department of Quantitative Methods and Economic Informatics. University of Zilina Žilina, Slovakia
philfv@hit.edu.cn, juneyoung9724@gmail.com,
jerrylin@ieee.org, jfrnda@gmail.com

Abstract. High utility itemset mining consists of identifying all the sets of items that appear together and yield a high profit in a customer transaction database. Recently, this problem was extended to discover trending high utility itemsets (itemsets that yield an increasing or decreasing profit over time). However, an important limitation of that problem is that it is assumed that trends remain stable over time. But in real-life, trends may change in different time intervals due to specific events. To identify time intervals where itemsets have increasing/decreasing trends in terms of utility, this paper proposes the problem of mining Locally Trending High Utility Itemsets (LTHUIs) and their Trending High Utility Periods (THUPs). Properties of the problem are studied and an efficient algorithm named LTHUI-Miner is proposed to enumerate all the LTHUIs and their THUPs. An experimental evaluation shows that the algorithm is efficient and can discover insightful patterns not found by previous algorithms.

Keywords: High Utility Mining · Trending Itemset · Local Trends.

1 Introduction

Frequent itemset mining (FIM) is a popular data mining task, which consists of enumerating all sets of values (items) that have a support (occurrence frequency) that is no less than a minimum threshold in a transaction database [5]. FIM has recently been generalized as high utility itemset mining (HUIM) to consider items having non binary purchase quantities in transactions and weights indicating their relative importance [2, 4, 10, 13–15]. The goal of HUIM is to find all itemsets that have a high utility (e.g. yield a high profit). Though, HUIM is useful to understand customer behavior, a key problem of HUIM is that the time dimension is ignored. But in real-life, the utility of itemsets vary over time. For

example, the sales of some products in a retail store may increase or decrease over a few weeks as it loses or gains in popularity.

To discover high utility itemsets that have an increasing or decreasing utility over time, the problem of mining trending HUIs was proposed [9]. However, this problem only focuses on discovering itemsets that have trends spanning over the whole database (e.g. a set of products having sales that always follows an upward or downward trend). But that assumption is often unrealistic as an itemset may have upward or downward trends only during some time periods rather than in the whole database. For instance, the utility (profit) generated by the sale of sunscreen in a store may have an upward trend from May to July but not during the whole year. It is thus an important challenge to design algorithms to identify trends in non predefined time intervals. This is also challenging as it requires to not only consider a large search space of itemsets but also of time intervals.

This paper addresses this issue by proposing a novel problem of mining locally trending high utility itemsets (LTHUIs), that is to find all time intervals where itemsets have a high utility and show an upward or downward trend. To efficiently discover these patterns, this paper proposes a novel algorithm named Locally Trending High Utility Itemset Miner (LTHUI-Miner). It relies on novel upper-bounds and pruning techniques. An experimental evaluation on real transaction data shows that the proposed algorithm has excellent performance and can discover insightful patterns not found by previous algorithms.

The rest of this paper is organized as follows. Section 2 reviews related work. Section 3 defines the proposed problem of LTHUI mining. Then, Section 4 describes the designed algorithm, Section 5 presents the experimental evaluation. Lastly, Section 6 draws a conclusion and discusses future work.

2 Related work

HUIM extends FIM [1, 5] and thus algorithms for these problems have similarities. However, there is also a key difference. FIM algorithms discover frequent itemsets by relying on the anti-monotonicity property of the support measure, which states that the support of an itemset cannot be greater than that of its subsets [1, 5]. This is a very powerful property to reduce the search space, but it does not hold for the utility measure in HUIM. To mine high utility itemsets efficiently, state-of-the-art HUIM algorithms such as Two-Phase [14], HUI-Miner [15], d2hup [13] and HU-FIMi [10] introduced various upper-bounds on the utility measure that respect the anti-monotonicity property to reduce the search space, and novel data structures to perform utility computation efficiently.

Though HUIM is useful to reveal profitable customer behavior, few HUIM algorithms consider the time dimension. The PHM algorithm [3] finds patterns that periodically appear and yield a high profit (e.g. a customer buys wine every week). The RUP algorithm [8] finds itemsets that recently had a high utility by applying a decay function to the utility measure (recent events are considered more important in utility calculations). And recently, to discover itemsets that follow some trends such as an increase or decrease in utility, the TPHUI-Miner

algorithm [9] was designed. However, a major limitation of the three above algorithms is that they find patterns that shows some periodic behavior, recent behavior or trends valid for the whole database rather than for specific time intervals. But in real-life, the utility of itemsets vary over time, and some of these behaviors may only appear in some time intervals.

To find itemsets that have a high utility in some specific time periods, on-shelf high utility itemset mining was proposed [11]. However, the time periods need to be fixed by the user beforehand. To find high utility itemsets in non predefined time intervals, it was proposed to mine local high utility itemsets with the LHUI-Miner algorithm [7]. Though this algorithm can find insightful patterns, it is unable to discover trends such as an increase or decrease of utility in specific time periods. To address these limitations, the next section proposes the novel problem of discovering locally trending high utility itemsets.

3 Problem definition

This section introduces HUIM, and then defines the proposed problem of mining locally trending high utility itemsets. The input of HUIM is a transaction database. Consider a set of items (products) $I = \{i_1, i_2, \ldots, i_n\}$. A subset $X \subseteq I$ is called an itemset. An itemset $\{i\}$ containing a single item i can be denoted without brackets as i, when the context is clear. A transaction T is an itemset, purchased by a customer. A *transactional database* is a multiset of transactions $D = \{T_1, T_2, \ldots, T_m\}$, where each transaction $T_{tid} \in D$ has a unique identifier tid and a timestamp $t(T_{tid})$, which may not be unique. Each item i appearing in a transaction T is associated with a number $q(i, T) \in \mathbb{N}^+$ called its internal utility (purchase quantity). Moreover, each item $i \in I$ is associated with an external utility value $p(i) \in \mathbb{N}^+$ representing its relative importance (e.g. unit profit). For instance, Table 1 shows a database containing five items (a, b, c, d, e) and nine transactions $(T_1, T_2, \ldots, T_9)$, which will be used as running example. Timestamps are denoted as $d_1, d_3, \ldots, d_{12}$. The internal utility of an item in a transaction is shown as a number besides the item, while the external utility of items is given in Table 2. Transaction T_1 indicates that a customer purchased the items b, c, and e with purchase quantities (internal utility) of 2, 2 and 1, respectively. Their external utility (unit profit) are 2, 1 and 3, respectively.

The task of HUIM consists of enumerating all high utility itemsets, i.e. itemsets having a utility that is no less than a positive minimum utility threshold (*minutil*) set by the user [14].The utility of an item i in a transaction T is defined as $u(i, T) = p(i) \times q(i, T)$. The utility of an itemset X in T is defined as $u(X, T) = \sum_{i \in X \wedge X \subseteq T} u(i, T)$ if $X \subseteq T$, and otherwise $u(X, T) = 0$. The utility of an itemset X in a database D is defined as $u(X) = \sum_{T \in D \wedge X \subseteq T} u(X, T)$. For example, the utility of itemset $\{a, c\}$ in the database is $u(\{a, c\}) = u(\{a, c\}, T_4) + u(\{a, c\}, T_5) + u(\{a, c\}, T_8) = 12 + 16 + 16 = 44$.

To find HUIs having increasing/decreasing trends in terms of utility in a database, Hackman et al [9] proposed to mine *trending high utility itemsets*, i.e. HUIs having a positive/negative slope for a whole database. The slope of a HUI

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	$(b,5),(c,1),(e,1)$	d_4
T_4	$(a,2),(b,10),(c,2)$	d_4
T_5	$(a,2),(c,6),(e,2)$	d_6
T_6	$(b,4),(c,3)$	d_7
T_7	$(b,16),(c,2)$	d_9
T_8	$(a,2),(c,6),(e,2)$	d_{10}
T_9	$(b,5),(c,2),(e,1)$	d_{12}

Table 2. External utilities of items

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

is defined as follows. The utility of an itemset X at a timestamp d in a database D is defined as: $u(X,d) = \sum_{X \subseteq T \in D \wedge t(T)=d} u(X,T)$. Let there be a HUI X and TS be the set of timestamps in a database D. The utility set of X in D is defined as the multiset $US(X) = \{u(X,d) | d \in TS\}$. The slope of X in D is: $slope(X,D) = \frac{\sum_{d \in TS}(u(X,d)-avg(US(X)) \times (d-avg(TS))}{\sum_{u \in US(X)}(u-avg(US(X))^2}$ where avg is the average.

There are two important issues with the problem of mining trending HUIs [9]. First, in the above slope calculation, it can be argued that time should be used as denominator instead of the utility because the user is typically interested in how utility varies over time rather than the opposite. Second, the slope of a HUI is calculated for the whole database. Hence, the algorithm of Hackman et al. [9] is unable to find local trends such as a HUI that follows a trend only in a sub-time interval. To address these issues, this paper proposes to mine itemsets that have a high utility and follow an increasing/decreasing trend in some non predefined time intervals. This paper redefines the concepts of utility and slope such that the time is divided into non-overlapping consecutive bins to reduce the influence of small fluctuations in the utility of items. The user must set a bin length $binlen \in \mathbb{Z}^+$. Then, the average timestamp and average utility of each bin is used as basis for slope calculations.

Definition 1 (Bin). *Let there be a database D of m transactions, and two timestamps i, j such that $0 \leq i \leq j$. The bin from time i to j is defined as $B_{i,j} = \{T | i \leq t(T) \leq j \wedge T \in D\}$. The length of a bin $B_{i,j}$ is length($B_{i,j}$) = j - i + 1. The average timestamp of a bin $B_{i,j}$ is defined as $at(B_{i,j}) = \frac{i+j}{2}$. The utility of an itemset X in a bin $B_{i,j}$ is defined as: $u(X,B_{i,j}) = \sum_{X \subseteq T \in B_{i,j}} u(X,T)$. The average utility of X in $B_{i,j}$ is defined as $au(X,B_{i,j}) = \frac{u(X,B_{i,j})}{length(B_{i,j})}$.*

Definition 2 (Binned database). *Let there be a database $D = \{T_1, T_2, \ldots, T_m\}$ and a fixed bin length binlen. The time interval $[t(T_1), t(T_m)]$ is divided into consecutive non-overlapping bins of length binlen. For the sake of simplicity, the last bin is ignored if its length is less than binlen. The number of bins in D is $numbin = \left\lfloor \frac{t(T_m)-t(T_1)}{binlen} \right\rfloor$. The sequence of bins in D, ordered by time, is defined as: $BS = \langle B_{1,binlen}, B_{binlen+1,binlen \times 2}, \ldots, B_{binlen \times (numbin-1)+1,binlen \times numbin} \rangle$. Moreover, let $BS[k]$ denotes the k-th element of BS.*

To detect non predefined time intervals containing trends, a sliding window of length *winlen* is slided over the sequence of bins BS.

Definition 3 (Window). *Let there be a database D and a user-defined sliding window length winlen, such that $winlen = binlen \times k$ where $k \in \mathbb{Z}$ and $k \geq 2$. Each window contains winlen/binlen bins. Let $W_{[i,j]}$ denotes the window containing the i-th bin until the j-th bin of the sequence BS, that is $W_{[i,j]} = \{BS[k] | i \leq k \leq j\}$. A window $W_{[k,l]}$ is a subset of $W_{[i,j]}$ iff $W_{[k,l]} \subseteq W_{[i,j]}$ $(i \leq k, l \leq j)$, i.e. all bins included in $W_{[k,l]}$ are also included in $W_{[i,j]}$. A window $W_{[k,l]}$ is a strict subset of $W_{[i,j]}$ iff $W_{[k,l]} \subset W_{[i,j]}$. The length of a window $W_{[i,j]}$ is length$(W_{[i,j]}) = binlen \times (j - i + 1)$. Let $BN_{[i,j]}$ be the sequence of bins that are contained in $W_{[i,j]}$, ordered by time, that is $BN_{[i,j]} = \langle BS[i], BS[i+1] \ldots, BS[j] \rangle$. Let $AU(X)_{[i,j]}$ denotes the sequence of average utilities of an itemset X for the bins of $BN_{[i,j]}$, that is $AU(X)_{[i,j]} = \langle au(X, BS[i]), au(X, BS[i+1]) \ldots, au(X, BS[j]) \rangle$. Let $AT_{[i,j]}$ denotes the sequence of average timestamps corresponding to bins in $BN_{[i,j]}$, that is $AT_{[i,j]} = \langle at(BS[i]), at(BS[i+1]) \ldots, at(BS[j]) \rangle$. In the following, indices $[i, j]$ of W, BN, AU, and AT (which refer to sequence BS) are omitted when the context is clear. The utility of an itemset X in a window W is defined as: $u(X, W) = \sum_{B \in W} u(B, X)$.*

We then define the slope of an itemset in a sliding window as follows:

Definition 4 (Slope of an itemset in a sliding window). *Let $A[k]$ be the k-th element of a sequence of values A. The slope of an itemset X in a sliding window W is: $slope(X, W) = \frac{\sum_{k=1 \ldots |BN|} (AU(X)[k] - avg(AU(X))) \times (AT[k] - avg(AT))}{\sum_{t \in AT} (t - avg(AT))^2}$ iff the itemset X appears in each bin of the sliding window W, i.e., $AU(X)[k] \neq 0$. A sliding window W meeting that latter condition is called a no-empty-bin sliding window of X. Otherwise, the slope is undefined. Besides, in the case where the denominator is 0, the slope is defined as 0.*

For example, if $binlen = 3$, $winlen = 2 \times binlen = 6$, $BS = \langle B_{1,3}, B_{4,6}, B_{7,9}, B_{10,12} \rangle$, and $W_{[1,2]} = \{B_{1,3}, B_{4,6}\}$. The utility of itemset $\{b, c\}$ in $B_{1,3}$ is $u(\{b, c\}, B_{1,3}) = u(\{b, c\}, T_1) + u(\{b, c\}, T_2) = 6 + 11 = 17$, the utility of itemset $\{b, c\}$ in $W_{[1,2]}$ is $u(\{b, c\}, W_{[1,2]}) = u(\{b, c\}, B_{1,3}) + u(\{b, c\}, B_{4,6}) = 17 + 33 = 50$. The slope of itemset $\{b, c\}$ in $W_{[1,2]}$ is $slope(\{b, c\}, W_{[1,2]}) = \frac{(5.67 - 8.34) \times (2 - 3.5) + (11 - 8.34) \times (5 - 3.5)}{(2 - 3.5)^2 + (5 - 3.5)^2} = 1.78$.

If *binlen* is set to a reasonably large value, the requirement that an itemset X appears in each bin of a sliding window to have a slope is reasonable, and ensures that the slope is not influenced by missing values. Based on the above definitions, the problem of mining locally trending HUIs is defined.

Definition 5 (Problem definition). *Let there be some parameters $binlen \in \mathbb{Z}^+$, $winlen = x \times binlen$ for an integer $x \in \mathbb{Z}^+$ such that $x \geq 2$, $minutil \geq 0$, $minslope > 0$ (or $maxslope < 0$) set by the user. A window $W_{[i,j]}$ is a Trending High Utility Period (THUP) of an itemset X if for any sliding window $W_{[k,l]} \subseteq W_{[i,j]}$ where length $(W_{[k,l]}) = winlen$, $u(X, W_{[k,l]}) \geq minutil$, $slope(X, W_{[k,l]}) \geq$*

minslope, indicating an increasing trend (or slope$(X, W_{[k,l]}) \leq$ *maxslope, indicating a decreasing trend). Furthermore, a THUP* $W_{[i,j]}$ *is said to be a maximum THUP if there is no THUP* $W_{[o,p]}$ *such that* $W_{[i,j]} \subset W_{[o,p]}$*. The problem of Locally Trending High Utility Itemset Mining (LTHUIM) is to find all Locally Trending High Utility Itemsets (LTHUIs), and their maximum Trending High Utility Periods (THUPs). An itemset is a LTHUI if it has at least one THUP.*

For example, for $binlen = 3, winlen = 6, minutil = 20$ and $minslope = 0.15$, three LTHUIs are found. $\{b\}$ has a maximum THUP $[d_1, d_9]$ (utility = 82, slope = 0.52), $\{b, c\}$ has a maximum THUP $[d_1, d_9]$ (utility = 95, slope = 0.52), and $\{c, e\}$ has a maximum THUP $[d_1, d_6]$ (utility = 27, slope = 0.19).

4 The LTHUI-Miner Algorithm

The search space in traditional HUIM consists of $2^I - 1$ itemsets. For the proposed problem, if there are w sliding windows, then there are $(2^I - 1) \times w$ potential THUPs to be considered. To efficiently find LTHUIs, the proposed LTHUI-Miner uses three properties that reduce the search space by eliminating items or itemsets w.r.t. the whole database or a sliding window.

Property 1 (Pruning a Low-TWU Item in a Database). For an item i and a database D, let there be a measure $TWU(i) = \sum_{T \in D \wedge i \in T} u(T, T)$. If $TWU(i) <$ $minutil$, then any itemset $X \ni i$ is not a LTHUI.

This property was proven for HUIs in the traditional HUIM problem [14]. But it also holds for LTHUIM since every LTHUI must be a HUI.

For example, if $minutil = 20$, $binlen = 3$ and $winlen = 6$, $TWU(d) = \sum_{T \in D \wedge \{d\} \in T} u(T, T) = u(T_2, T_2) = 18 < minutil$. Thus, d is a low TWU item in the database, and any itemset $X \ni \{d\}$ is not a LTHUI.

The second and third pruning properties require a total order $\prec$ on the set of items I, which is used by LTHUI-Miner to explore the search space of itemsets. LTHUI-Miner performs a depth-first search starting from itemsets containing single items, and recursively extends each itemset by appending single items according to that order. Formally, an itemset $X \cup \{y\}$ obtained by appending an item y to an itemset X is said to be an extension of X if $i \prec y$, $\forall i \in X$.

Property 2 (Pruning an Unpromising itemset using its Remaining Utility in a Database). The remaining utility of an itemset X in a transaction T is defined as $ru(X, T) = \sum_{i \in T \wedge i \succ x \forall x \in X} u(i, T)$ if $X \subseteq T$. The *remaining utility of an itemset* X *in a database* is defined as $reu(X) = \sum_{T \in D \wedge X \subseteq T} ru(X, T)$. If $u(X) +$ $reu(X) < minutil$, then X and its transitive extensions are not LTHUIs.

For example, if $minutil = 30$, $binlen = 3$ and $winlen = 6$. The TWU ascending order on items is $a \prec e \prec b \prec c$. Note that item d has been pruned using Property 1. $u(\{c\}) + reu(\{c\}) = 27 + 0 < minutil$. Then, itemset $\{c\}$ and its transitive extensions are not LTHUIs.

Property 3 (Pruning an Unpromising itemset using its Remaining Utility in a sliding window). The *remaining utility of an itemset X in a sliding Window W* is defined as $reu(X, W) = \sum_{T \in B \in W \wedge X \subseteq T} ru(X, T)$. If $u(X, W) + reu(X, W) < minutil$, then X and its transitive extensions have no THUP in W.

This property can be proved by observing that such itemsets cannot have a utility greater than or equal to *minutil* in the sliding window W, and thus these itemsets cannot have a THUP in W. For example, if $minutil = 20$, $binlen = 3$ and $winlen = 6$, $u(\{e\}, W_{[1,2]}) + reu(\{e\}, W_{[1,2]}) = 6 + 9 = 15 < minutil$. Thus, the window $W_{[1,2]}$ is not a THUP for itemset $\{e\}$ and its transitive extensions.

To efficiently calculate the utility of any itemset during the depth-first search and check the pruning conditions of Properties 2 and 3, the proposed algorithm utilizes a novel structure called *Trending Utility-list* (TU-list), which extends the utility-list structure used in traditional HUIM [4] with information about bins and time periods. The first part of a TU-list of an itemset X stores information about the utility of the itemset X in transactions where it appears, and about the utilities of items that could extend X in these transactions. Formally, the first part of a TU-list is a set of tuples called *elements* such that there is a tuple $(tid, iutil, rutil)$ for each transaction T_{tid} containing X where $iutil = u(X, T_{tid})$ and $rutil = ru(X, T_{tid})$. The second part of a TU-list contains four lists named $binUtils$, $binRutils$, $trendPeriods$ and $promisingPeriods$. They store the utility of X for each bin, the remaining utility of X for each bin, the maximum trending high utility periods of X and the promising periods of X, respectively. A *promising period* of an itemset X is a time period where X and its transitive extensions may have a utility greater than or equal to *minutil* based on Property 3. Formally, let there be some parameters $winlen \in \mathbb{Z}^+$ and $minutil \geq 0$ set by the user. A window $W_{[i,j]}$ is a *promising period* of an itemset X if for any sliding window $W_{[k,l]} \subseteq W_{[i,j]}$ where length $(W_{[k,l]}) = winlen$, $u(X, W_{[k,l]}) + reu(X, W_{[k,l]}) \geq minutil$.

The TU-list structure of an itemset X has two interesting properties. First, it allows to directly calculate the utility $u(X)$ of X without scanning the database, as the sum of the *iutil* values in the TU-list of X. Second, $reu(X)$ can be calculated as the sum of *rutil* values. Moreover, the utility and remaining utility of an itemset X in a bin B and a window W can also be calculated from its TU-list by considering only transactions in B and W, respectively.

For example, the TU-list of itemset $\{e\}$ is $elements = \langle(0, 3, 6), (1, 3, 11), (2, 3, 11), (4, 6, 6), (7, 6, 6), (8, 3, 12)\rangle$, $binUtils = \langle 6, 9, 0, 9\rangle$, $binRutils = \langle 17, 17, 0, 18\rangle$, $trendPeriods = \langle\rangle$ and $promisingPeriods = \langle W_{[1,2]}\rangle$. Then, the utility of itemset $\{e\}$ in a database or window can be calculated without scanning the database again, e.g., $u(\{e\}) = 3 + 3 + 3 + 6 + 6 + 3 = 24$, $u(\{e\}, W_{[1,2]}) = binUtils[1] + binUtils[2] = 6 + 9 = 15$. The remaining utility of itemset $\{e\}$ in a window can also be calculated directly using $binRutils$: $reu(\{e\}, W_{[1,2]}) = binRutils[1] + binRutils[2] = 17 + 17 = 34$.

Another property of TU-lists is that those of two itemsets of the form $P \cup \{x\}$ and $P \cup \{y\}$ can be joined to obtain the TU-list of an itemset $P \cup \{x, y\}$. This is done by first applying the construct procedure of HUI-Miner [15]. Then, the

binUtils, *binRutils*, *trendPeriods* and *promisingPeriods* lists can be calculated by applying the *findTrend* procedure, presented in the next section.

The algorithm. We next present the proposed LTHUI-Miner algorithm by explaining how it finds increasing trends. Decreasing trends are found in a similar way. The algorithm takes as input a transaction database D and the *binlen*, *winlen*, *minutil* and *minslope* parameters. The algorithm outputs all LTHUIs and their maximum THUPs. The algorithm first scans the database to calculate the bins, sliding windows and $TWU(i)$ of each item i. Then, each item i such that $TWU(i) < minutil$ is ignored from further processing as it cannot be part of a LTHUI by Property 1. Then, the processing order $\prec$ on remaining items is defined as the increasing order of TWU, as in previous work [4]. Then, the algorithm scans the database again to create the TU-list of each remaining item. Thereafter, LTHUI-Miner recursively extends each of those items by appending items following the $\succ$ order. This is done by calling the LTHUISearch procedure (Algorithm 1) with six parameters: (1) an itemset P (initially $P = \emptyset$), (2) a set exP of one-item extensions of P of the form $Px = P \cup \{x\}$ where $x \in I$ (initially, the remaining items), (3) *binlen*, (4) *winlen*, (5) *minutil*, and (6) *minslope*. The procedure first checks if the *trendPeriods* list of each itemset Px in the set exP is empty. If not, the itemset Px is output as a $LTHUI$ with $Px.TUlist.trendPeriods$ as its maximum THUPs. Moreover, if *promisingPeriods* of Px is not empty and Px is promising in the database according to Property 2, the algorithm will try to extend Px. This is done by joining Px with each itemset $Py \in exP$ such that $y \succ x$, to obtain itemsets of the form Pxy. The TU-list of Pxy is constructed by calling the *construct* procedure. Then, the procedure $FindTrend$ is called to construct the *binUtils*, *binRutils*, *trendPeriods* and *promisingPeriods* of that TU-list, and Pxy is added to a set $exPx$. Then, the procedure *LTHUI-Search* is called with Px and $exPx$ to check if itemsets in $exPx$ are $LTHUIs$ and explore their extensions.

The $FindTrend$ procedure takes as input (1) an itemset P, (2) a one item extension of P, (3) *binlen*, (4) *winlen*, (5) *minutil* and (6) *minslope*. First, the procedure scans the *elements* of the TU-list of Px to calculate *binUtils* and *binRutils*. Then, the procedure moves a sliding window over the sequence of bins BS to calculate the utility and slope of windows using two variables, namely *winStart* (the index in BS of the first bin of a sliding window, initialized to 0) and *winEnd* (the index in BS of the last bin of a sliding window, initialized to *winlen*/*binlen*). However, the process of sliding a window while calculating the slope and utility may be interrupted because some sliding windows in BS may have empty bins, and the slope cannot be calculated in that case. Thus, a loop is performed to find the next sliding window without empty bins, and then continue the sliding process until an empty bin is encountered or *winEnd* reaches the last bin of the sequence BS. In more details, this is done by first finding the first no-empty-bin sliding window starting from *winStart*, updating *winStart*, *winEnd* and calculating *utils* (utility of the itemset Px in that window), *rutils* (remaining utility of Px in that window) . Then, the following step is repeated until $(winEnd + 1)$ reaches the last bin of BS or the utility of Px in the bin

of index $(winEnd + 1)$ is 0 or itemset P is unpromising in the sliding window $W_{[winStart+1,winEnd+1]}$: (1) increase the index of the first and last bin of the sliding window, then update $utils$ and $rutils$, (2) compare the value of $utils$, $utils + rutils$ with $minutil$ to determine whether to merge the sliding window with the previous period or add that window to $Px.TUlist.trendPeriods$ and $Px.TUlist.promisingPeriods$. These latter are used to store maximum THUPs and promising periods.

LTHUI-Miner is correct and complete, as it explores itemsets by recursively performing extensions of single items, and the algorithm only prunes extensions based on the pruning properties.

Algorithm 1: LTHUISearch

input : P: an itemset, exP: a set of one item extensions of itemset P, $binlen$: the length of a bin, $winlen$: the length of a sliding window, $minutil, minslope$: the minimum utility and slope thresholds.
output: the LTHUIs that are transitive extensions of P and their maximum THUPs

```
1  foreach itemset Px in exP do
2      if Px.TUlist.trendPeriods ≠ ∅ then output Px with Px.TUlist.trendPeriods;
3      if Px.TUlist.promisingPeriods ≠ ∅ and
         Px.TUList.sumIUtils + Px.TUList.sumRUtils ≥ minutil then
4          exPx ← ∅;
5          foreach itemset Py such that y ≻ x in exP do
6              Pxy ← construct(P, Px, Py);
7              FindTrend (Px, Pxy, binlen, winlen, minutil, minslope);
8              exPx ← exPx ∪ {Pxy};
9          end
10     end
11     LTHUISearch (Px, exPx, binlen, winlen, minutil, minslope);
12 end
```

5 Experiment

To test the performance of LTHUI-Miner, experiments were done on a computer having an Intel Xeon E3-1270 v5 processor with 64 GB RAM, on Windows 10. LTHUI-Miner was implemented in Java. Two real-life datasets with timestamps were used: $retail$ and $foodmart$. Let $|I|$, $|D|$ and A represents the number of distinct items, the number of transactions and the average transaction length. $retail$ contains transactions from an anonymous Belgian retail store ($|I|$ = 16,470, $|D|$ = 88,162, A = 10.30). $foodmart$ is transactional data obtained and transformed from the SQL-Server 2000 distribution ($|I|$ = 1559, $|D|$ = 4141, A = 4.40). The timestamps of $retail$ and $foodmart$ were generated by adopting a distribution used in prior work for retail data [7].

Because LTHUIM is a new problem, the performance of LTHUI-Miner cannot be compared with prior work. Thus, we compared three versions of LTHUI-Miner: (1) LTHUI-Miner (with all pruning techniques), denoted as *lthui*, (2) LTHUI-Miner without Property 3, denoted as *lthui-no-prop*3, and (3) a version

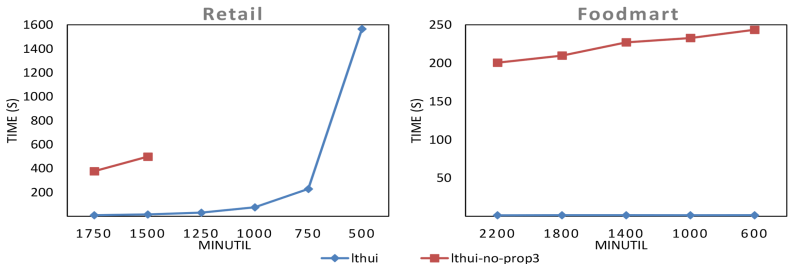
Algorithm 2: FindTrend

input : P: an itemset, Px: a one item extension of P, $binlen$: the length of a bin, $winlen$: the sliding window length, $minutil$: the minimum utility threshold, and $minslope$: the minimum slope threshold.
output: Calculate the $binUtils$, $binRutils$, $trendPeriods$ and $promisingPeriods$ of the TU-list of Px

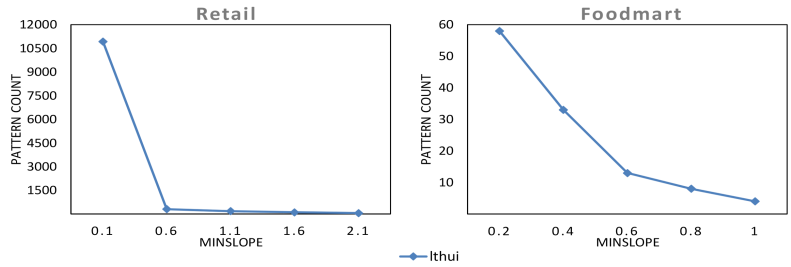
```
1  Scan the elements of the TU-list of Px to calculate binUtils and binRutils;
2  numBPW = winlen/binlen (the number of bins per sliding window);
3  winStart = 0 (the index of a sliding window's first bin in BS);
4  winEnd = numBPW (the index of a sliding window's last bin in BS);
5  while winEnd < BS.size do
6      Find the first no-empty-bin sliding window W for Px starting from winStart;
7      Update winStart and winEnd in terms of W;
8      Calculate utils = u(Px, W) and rutils = reu(Px, W);
9      while winEnd + 1 < BS.size and Px.TUlist.binUtils.get(winEnd + 1) ≠ 0 and P
         is promising in W_[winStart+1,winEnd+1] do
10         // increase index of the first bin of the sliding window
11         utils = utils − Px.binUtils.get(winStart);
12         rutils = rutils − Px.binRutils.get(winStart);
13         winStart = winStart + 1;
14         // increase index of the last bin of the sliding window
15         utils = utils + Px.binUtils.get(winEnd);
16         rutils = rutils + Px.binRutils.get(winEnd);
17         winEnd = winEnd + 1;
18         Merge the [winStart, winEnd] period with the previous trend period if
             utils ≥ minutil and slope(Px, W_[winStart,winEnd]) ≥ minslope. Otherwise add
             it to Px.TUlist.trendPeriods.
19         Merge the [winStart, winEnd] period with the previous promising period if
             utils + rutils ≥ minutil. Otherwise add it to Px.TUlist.promisingPeriods.
20     end
21 end
```

of LTHUI-Miner without Property 2 and 3. However, that latter ran out of memory for all the experiments, and thus its results are not reported in the following. Experiments were done by varying the *minutil* and *minslope* parameters to see the influence on runtime and pattern count, respectively. No results are shown for an algorithm if it ran out of memory, or the runtime exceeded one hour.

Influence of *minutil* on runtime and memory. In the first experiment, the parameter *minutil* was varied to evaluate the performance of LTHUI-Miner in terms of runtime. LTHUI-Miner was run with $winlen = 2000$ (about 5.5 hours), $binlen = 1000$ and $minslope = 0.1$ on the *retail* dataset, and run with $winlen = 500$, $binlen = 250$ and $minslope = 0.1$ on the *foodmart* dataset. Figure 1 (*a*) compares the runtimes of *lthui* and *lthui-no-prop*3 for the two datasets. It is observed that as *minutil* is decreased, runtime increases, which is reasonable since more patterns may be found. It is also observed that pruning an unpromising itemset in a sliding window using the remaining utility (Property 3) greatly reduces the runtime. For example, on the *retail* dataset, when $minutil = 1500$, the execution time of *lthui-no-prop3* is 498 s, which is more than 32 times that of *lthui*, and on the *foodmart* dataset, when $minutil = 1400$, *lthui* is up to 176 times faster than *lthui-no-prop3*. Memory consumption was also measured to compare the two algorithm versions. It was found that in most cases, the memory usage of *lthui* is less than *lthui-no-prop3*, which shows that Property 3 reduces memory consumption. Details are not shown due to the page limitation.



(a) Influence of *minutil* on runtime



(b) Influence of *minslope* on the number of patterns found

Fig. 1. Experiment Results

Influence of *minslope* on the number of patterns found. In the second experiment, the *minslope* parameter was varied to evaluate its influence on the number of patterns found. Algorithms were run with $binlen = 1000$, $winlen = 2 \times binlen$ and $minutil = 600$ on the *retail* dataset and $binlen = 250$, $winlen = 2 \times binlen$ and $minutil = 100$ on the *foodmart* dataset. Results for the number of patterns are shown in Figure 1 (*b*) for the two datasets. It is observed that as *minslope* increases, the number of patterns decreases, which was expected.

Pattern analysis. On the two datasets, some patterns having a strong trend were found, which means that the utility of these itemsets was high and increased rapidly in their THUPs. For example, on *retail* and *foodmart* dataset, 179 and 13 patterns have slope values greater than 1.1 and 0.6 respectively. Discovering such strong trends can be very helpful for a retail store manager to understand customer behavior and take decisions, since products in LTHUIs generate high profits and the profits is growing quickly during their THUPs.

6 Conclusion

This paper has defined a novel problem of mining Locally Trending High Utility Itemsets having increasing/decreasing trend(s) in some non-predefined time periods. The properties of LTHUI mining were studied and a novel algorithm named LTHUI-Miner was proposed to efficiently mine all LTHUIs and their maximum THUPs. Besides, three pruning strategies were designed to improve the

performance of LTHUI-Miner. The experimental evaluation has shown that the algorithm is efficient and can find useful patterns. In future work, techniques to automatically adjust parameters will be considered, as well as extensions for high utility episode mining [6], incremental pattern mining [12, **?**] and using swarm optimization [16].

References

1. Agrawal, R., Srikant, R., et al.: Fast algorithms for mining association rules. In: Proc. of 20th int. conf. very large data bases, VLDB. vol. 1215, pp. 487–499 (1994)
2. Dawar, S., Goyal, V.: Up-hist tree: An efficient data structure for mining high utility patterns from transaction databases. In: Proc. 19th Intern. Conf. Database Engineering & Applications Symposium. pp. 56–61 (2015)
3. Fournier-Viger, P., Lin, C.W., Duong, Q.H., Dam, T.L.: Phm: Mining periodic high-utility itemsets. In: Proc. 16th Indust. Conf. Data Mining. pp. 64–79 (2016)
4. Fournier-Viger, P., Lin, J.C.W., Truong-Chi, T., Nkambou, R.: A survey of high utility itemset mining. In: High-Utility Pattern Mining, pp. 1–45. Springer (2019)
5. Fournier-Viger, P., Lin, J.C.W., Vo, B., Chi, T.T., Zhang, J., Le, B.: A survey of itemset mining. WIREs Data Mining and Knowledge Discovery (2017)
6. Fournier-Viger, P., Yang, P., Lin, J.C.W., Yun, U.: Hue-span: Fast high utility episode mining. In: Proc. 14th Intern. Conference on Advanced Data Mining and Applications. pp. 169–184 (2019)
7. Fournier-Viger, P., Zhang, Y., Lin, C.W., Fujita, H., Koh, Y.S.: Mining local and peak high utility itemsets. Inf. Sci. **481**, 344–367 (2019)
8. Gan, W., Lin, C.W., Fournier-Viger, P., Chao, H.: Mining recent high-utility patterns from temporal databases with time-sensitive constraint. In: Proc. 18th Intern. Conf. on Data Warehousing and Knowledge Discovery. pp. 3–16 (2016)
9. Hackman, A., Huang, Y., Tseng, V.S.: Mining trending high utility itemsets from temporal transaction databases. In: Proc. 29th Intern. Conf. on Database and Expert Systems Applications. pp. 461–470 (2018)
10. Kiran, R.U., Reddy, T.Y., Fournier-Viger, P., Toyoda, M., Reddy, P.K., Kitsuregawa, M.: Efficiently finding high utility-frequent itemsets using cutoff and suffix utility. In: Proc. 23rd Pacific-Asia Conf. Advances in Knowledge Discovery and Data Mining. pp. 191–203 (2019)
11. Lan, G., Hong, T., Tseng, V.S.: Discovery of high utility itemsets from on-shelf time periods of products. Expert Syst. Appl. **38**(5), 5851–5857 (2011)
12. Lee, J., Yun, U., Lee, G., Yoon, E.: Efficient incremental high utility pattern mining based on pre-large concept. Eng. Appl. of AI **72**, 111–123 (2018)
13. Liu, J., Wang, K., Fung, B.C.: Direct discovery of high utility itemsets without candidate generation. In: Proc. 12th IEEE Intern. Conf. Data Mining. pp. 984–989. IEEE (2012)
14. Liu, Y., keng Liao, W., Choudhary, A.N.: A two-phase algorithm for fast discovery of high utility itemsets. In: Proc. 9th Pacific-Asia Conf. on Knowledge Discovery and Data Mining. p. 689–695 (2005)
15. Qu, J.F., Liu, M., Fournier-Viger, P.: Efficient algorithms for high utility itemset mining without candidate generation. In: High-Utility Pattern Mining, pp. 131–160. Springer (2019)
16. Song, W., Huang, C.: Discovering high utility itemsets based on the artificial bee colony algorithm. In: Proc. 22nd Pacific-Asia Conf. Advances in Knowledge Discovery and Data Mining. pp. 3–14 (2018)