Mining Local High Utility Itemsets

Philippe Fournier-Viger[1(✉)], Yimin Zhang[2], Jerry Chun-Wei Lin[3], Hamido Fujita[4], and Yun Sing Koh[5]

[1] School of Natural Sciences and Humanities, Harbin Institute of Technology, Shenzhen, China
philfv8@yahoo.com
[2] School of Computer Sciences and Technology, Harbin Institute of Technology, Shenzhen, China
mrzhangym@126.com
[3] Department of Computing, Mathematics and Physics, Western Norway University of Applied Sciences (HVL), Bergen, Norway
jerrylin@ieee.org
[4] Faculty of Software and Information Science, Iwate Prefectural University, Takizawa, Iwate, Japan
HFujita-799@acm.org
[5] Department of Computer Sciences, University of Auckland, Auckland, New Zealand
ykoh@cs.auckland.ac.nz

Abstract. High Utility Itemset Mining (HUIM) is the task of analyzing customer transactions to find the sets of items that yield a high utility (e.g. profit). A major limitation of traditional HUIM algorithms is that they do not consider that the utility of itemsets vary over time. Thus, traditional HUIM algorithms cannot find itemsets that have a high utility during specific time periods. This paper addresses this limitation by defining the problem of mining *local high utility itemsets* (LHUI). An efficient algorithms named LHUI-Miner is proposed to mine these patterns. Experimental results show that the proposed algorithms are highly-efficient and can find useful patterns not found by traditional HUIM algorithms.

Keywords: High-utility pattern mining · Local high-utility itemsets

1 Introduction

Frequent Itemset Mining (FIM) [1,2] is a fundamental data mining task, which consists of finding itemsets (sets of items) that are frequently purchased in a customer transaction database. 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) [3–6] has recently emerged as an important data mining task. It consists of finding itemsets (sets of items) that yield a high utility (e.g. importance or profit) in customer transaction databases. An itemset is a High Utility Itemset (HUI) in a database if its utility is

S. Hartmann et al. (Eds.): DEXA 2018, LNCS 11030, pp. 450–460, 2018.
https://doi.org/10.1007/978-3-319-98812-2_41

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. HUIM algorithms such as Two-Phase [5] have thus introduced upper-bounds on the utility measure such as the TWU, which is anti-monotonic, to reduce the search space. Several more efficient algorithms have then been proposed, such as UP-Growth, HUI-Miner and FHM [3,4,6].

Though, HUIM has many applications such as click stream analysis, market basket analysis and biomedical applications [4,6], a major limitation of HUIM is that 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. A few extensions of HUIM and FIM consider time. For example, *periodic high-utility itemset mining* [8] discovers itemsets that are periodically bought by customers and yield a high 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 over long periods of time. Another related work is *High On-shelf Utility itemset Mining* (HOUM) [10], 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 predefined time periods (e.g. *winter*), which is unrealistic because many products may have on-shelf time periods that may 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 [7,9]. However, it also only consider the relative order of transactions instead of their real time stamps. In other words, it 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. An efficient algorithm called LHUI-Miner is designed to discover LHUIs. It relies on a novel data structure named LU-list.

The rest of this paper is organized as follows. Section 2 introduces preliminaries and defines the problems of mining LHUIs. Section 3 presents the proposed algorithm. Section 4 presents the experimental evaluation. Lastly, Sect. 5 draws the conclusion.

2 Preliminaries and Problem Statement

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. For example, consider the database of Table 1, which will be used as 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 2. In this example, timestamps of transactions $T_1, T_2 \ldots T_8$ 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.

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	$(b,2),(c,2),(e,1)$	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

Definition 1 (Utility of an item/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 $u(X) = \sum_{T \in D \wedge X \subseteq T} u(X, T)$ [5].

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 $u(\{b, c\}) = u(\{b, c\}, T_1) + u(\{b, c\}, T_2) + u(\{b, c\}, T_4) + u(\{b, c\}, T_6) = 6 + 11 + 6 + 11 = 34$.

An itemset X is said to be a *high utility itemset* (HUI) if its utility $u(X)$ is no less than a user-specified positive threshold *minutil* [5]. The utility measure is not anti-monotonic. Thus pruning strategies used in FIM cannot be directly used in HUIM. To reduce the search space in HUIM, the Transaction Weighted Utilization (TWU) upper-bound was introduced [5].

Definition 2 (Transaction weighted utilization). The utility of a transaction T is defined as $tu(T) = \sum_{i \in T} u(i, T)$. The TWU of an itemset X is the sum of the utilities of transactions containing X, i.e. $TWU(X) = \sum_{X \subseteq T \wedge T \in D} tu(T)$.

Property 1. For two itemsets $X \subseteq X'$, $u(X') \leq TWU(X') \leq TWU(X)$ [5].

Thus, for an itemset X, $TWU(X)$ is an upper-bound on $u(X)$, and the TWU is anti-monotonic. Hence, if $TWU(X) < minutil$, itemset X and all its supersets can be pruned. A major limitation of HUIM is that it cannot find itemsets that yield a high utility in specific time periods but not in the whole database. For example, for $minutil = 50$, itemset $\{a, c\}$ is a HUI since $u(\{a, c\}) = 56 > 50$, and $\{d, e\}$ is not a HUI since $u(\{d, e\}) = 36 < minutil$. But from time $d5$ to $d6$, $\{d, e\}$ yields a utility that is more than twice the utility of $\{a, c\}$ (a utility of 26 instead of 12). Thus, during this period, $\{d, e\}$ is more interesting than $\{a, c\}$, but is ignored in HUIM. To address this problem, we propose a new type of patterns called *Local High utility itemsets* (LHUIs), defined as follows.

Definition 3 (Window). A window denoted as $W_{i,j}$ is the set of transactions from time i to j, i.e. $W_{i,j} = \{T | i \leq t(T) \leq j \wedge T \in D\}$, where i, j are integers. The length of a window $W_{i,j}$ is defined as $length(W_{i,j}) = j - i + 1$. The length of a database D containing m transactions is $W_D = t(T_m) - t(T_1) + 1$. A window $W_{k,l}$ is said to **subsume** another window $W_{i,j}$ iff $W_{i,j} \subsetneqq W_{k,l}$.

Definition 4 (Local high utility itemset). The utility of an itemset X in a window $W_{i,j}$ is defined as $u_{i,j}(X) = \sum_{T \in W_{i,j} \wedge X \subseteq T} u(X, T)$. An itemset X is a local high utility itemset (LHUI) if there exists a window $W_{i,j}$ such that $length(W_{i,j}) = minLength$ and $u_{i,j}(X) \geq lMinutil$, where $minLength \leq W_D$ and $lMinutil > 0$ are user-specified thresholds representing a minimum length and utility, respectively. Moreover, let $LHUIs$ denotes the set of all LHUIs.

For example, for $minLength = 3$ and $lMinutil = 20$, itemset $\{b, c\}$ is a LHUI since for the window W_{d_1,d_3} the utility of $\{b, c\}$ is $u_{d_1,d_3}(\{b, c\}) = u(\{b, c\}, T_1) + u(\{b, c\}, T_2) + u(\{b, c\}, T_3) = 6 + 11 + 6 = 23 \geq lMinutil = 20$, and $length(W_{d_1,d_3}) = 3 - 1 + 1 = 3 = minLength$.

Theorem 1 (Relationship between LHUIs and HUIs). *If* $minutil = lMinutil \times \lceil \frac{W_D}{minLength} \rceil$, *then* $HUIs \subseteq LHUIs$.

Proof. If Theorem 1 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 $u(X) = \sum_{i=0}^{\lceil \frac{W_D}{minLength} \rceil} WU_i < lMinutil \times \lceil \frac{W_D}{minLength} \rceil = minutil$. There is a contradiction between $u(X) < minutil$ and $X \in HUIs$. Thus, the Theorem 1 is proven. □

Definition 5 (LHUI period). For an itemset X, a window $W_{i,j}$ is a *LHUI period* if for each window $W_{k,l} \subseteq W_{i,j}$ of length $minLength$, $u_{k,l}(X) \geq lMinutil$. A LHUI period $W_{i,j}$ is said to be a *maximum LHUI period* if there is no LHUI period $W_{o,p}$ such that $W_{i,j} \subset W_{o,p}$.

For example, consider that $minLength = 5$ and $lMinutil = 30$. The window W_{d_1,d_6} is a LHUI period of $\{a, b, c\}$, because W_{d_1,d_5} and W_{d_2,d_6} are its two sub-windows of length $minLength$, and $u_{d_1,d_5}(\{a, b, c\}) = u_{d_2,d_6}(\{a, b, c\}) = 32 \geq lMinutil$. The maximum LHUI period of itemset $\{a, b, c\}$ is W_{d_1,d_9}.

Definition 6 (Local High Utility Itemset mining). The problem of Local High Utility Itemset Mining (LHUIM) is to find all LHUIs and their maximum LHUI periods given the parameters $minLength$ and $lMinutil$.

For example, given the database of Table 1, $minLength = 5$ and $lMinutil = 30$, 25 LHUIs are found with their maximum LHUI periods, including $\{a, d\}$:$[d_5, d_9]$, $\{a, b, c\}$:$[d_5, d_5]$, $\{c, e\}$:$[d_3, d_7]$, $\{b, e\}$:$[d_1, d_7]$, $\{a, c, e\}$:$[d_5, d_{10}]$, $\{b, d, e\}$:$[d_3, d_5]$, $\{b, c, e\}$:$[d_1, d_7]$, $\{b, c, d, e\}$:$[d_3, d_5]$. For an itemset X, the numbers between brackets indicate the first and last timestamps of transactions in the LHUI period of X that contains the itemset. This notation is called the *abbreviated LHUI period of* X. For example, although the LHUI period of $\{a, b, c\}$ is $[d_1, d_9]$, it only appears in d_5. Thus, its LHUI period is denoted as $[d_5, d_5]$. For $minutil = lMinutil \times \lceil \frac{W_D}{minLength} \rceil = 60$, traditional HUIM algorithms find 5 HUIs: $\{b, e\}$, $\{a, c, e\}$, $\{b, d, e\}$, $\{b, c, d, e\}$ and $\{b, c, e\}$. Here, all HUIs are LHUIs, and these latter provides LHUI period information.

3 Proposed Algorithm

This section presents an algorithm to efficiently mine LHUIs, named LHUI-Miner, which extends the HUI-Miner [4] algorithm. The algorithm is designed to explore the search space of itemsets by following a total order $\succ$ on items in I. It is said that an itemset Y is an *extension* of an itemset X if $Y = X \cup \{j\} \wedge \forall i \in X,\ j \succ i$. The proposed algorithms utilize a novel data structure called *Local Utility-list* (LU-list) to store information about each itemset, which adapts the utility-list [4] structure to store period information. The algorithm first scan the database to create a LU-list for each item. Then, it explores 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 allow to determine if an itemset is a LHUI without scanning the database. The LU-list structure is defined as follows.

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 $\sum_{i \in T_{tid} \wedge \forall j \in X, i \succ j} u(i, T_{tid})$ [4]. Moreover, the LU-list contains two sets named $iutilPeriods$ and $utilPeriods$, which stores the abbreviated maximum LHUI periods and PLHUI periods of X, respectively. The PLHUI periods of X are the periods where X and its extensions could be LHUIs.

Definition 7 (PLHUI period). The remaining utility of an itemset X in a window $W_{k,l}$ is defined as $ru_{k,l}(X) = \sum_{i \in T \wedge T \in W_{k,l} \wedge X \subseteq T \wedge \forall j \in X, i \succ j} u(i,T)$. For an itemset X, a window $W_{k,l}$ is a *PLHUI period* (promising LHUI period) if for each window $W_{y,z} \subseteq W_{k,l}$ of length $minLength$, $u_{y,z}(X) + ru_{y,z}(X) \geq lMinutil$.

For example, consider that $a \prec b \prec c \prec d \prec e$, $minLength = 3$ *days* and $lMinutil = 22$. The LU-list of $\{a\}$ is $\{(T_4, 10, 48), (T_5, 10, 12), (T_7, 10, 6), (T_8, 10, 12)\}$, $iutilPeriods = \emptyset$, and $utilPeriods = \{[d_5, d_6], [d_9, d_{10}]\}$. The LU-list of $\{d\}$ is $\{(T_2, 4, 3), (T_4, 20, 6), (T_7, 4, 0)\}$, $iutilPeriods = \{[d_3, d_5]\}$, and $utilPeriods = \{[d_3, d_5]\}$. And the LU-list of $\{a, d\}$ is $\{(T_4, 30, 6), (T_7, 14, 0)\}$, $iutilPeriods = \{[d_5, d_5]\}$, and $utilPeriods = \{[d_5, d_5]\}$. The LU-list of an itemset provides useful information. If $iutilPeriods$ is not empty, then X is a LHUI. And if $utilPeriods$ is empty, then it can be proven that all its extensions and transitive extensions cannot be LHUIs or PHUIs and can be pruned from the search space. The proof of this property is omitted due to space limitation.

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 $t(T_i)$. 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 [3]. 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 1) 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. It can be easily seen that this procedure is correct and complete to discover all LHUIs.

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 variable $winStart$ (initialized to 0; the first element of lUl), and $winEnd$. The procedure first scan lUl to find $winEnd$ (the end index of the

Algorithm 1. LHUI-Search

input : P: an itemset, $ExtensionsOfP$: 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 ∈ ExtensionsOfP do
2      if Px.LUList.iutilPeriods ≠ ∅ then output Px with Px.LUList.iutilPeriods;
3      if Px.LUList.utilPeriods ≠ ∅ then
4          ExtensionsOfPx ← ∅;
5          foreach itemset Py ∈ ExtensionsOfP such that y ≻ x do
6              Pxy.LUList ← Construct (P, Px, Py);
7              generatePeriods (Pxy, lMinutil, minLength);
8              ExtensionsOfPx ← ExtensionsOfPx ∪ Pxy;
9          end
10         LHUI-Miner (Px, ExtensionsOfPx, minutil, minLength);
11     end
12 end
```

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 $lMinutil$ 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.

Algorithm 2. The generatePeriods procedure

input : lUl: a LU-list, $lMinutil$: a user-specified utility threshold, $minLength$: a user-specified window length threshold

```
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
```

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

Strategy 1. Discarding unpromising items using the sliding window. A modified version of Property 1 is used in LHUI-Miner to discard items that cannot be in a LHUI or PHUI. The TWU of an item i in a window $W_{k,l}$ is defined as $TWU_{k,l}(i) = \sum_{i \in T \wedge T \in W_{k,l}} tu(T)$. 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.

Strategy 2. Discarding unpromising transactions using the sliding window. If for each item i in a transaction T, $TWU_{k,l}(i) < lMinutil$ for each window $W_{k,l}$ of length $minLength$ containing T, then the transaction T is discarded because this transaction cannot be in any LHUI period.

Strategy 3. Discarding unpromising tuples in each LU-list. The LU-list of an itemset X can stores numerous tuples that represents the transactions where X appears. However, some of those transactions are not in any PLHUI periods. Thus, these transactions are not in the LHUI periods of X and those of its transitive extensions. Hence, this strategy does not store the tuples representing these transactions in the LU-list of each itemset X. This reduces the runtime of the algorithms since performing the intersection of LU-lists and scanning LU-lists is faster for smaller LU-lists.

Strategy 1 and 2 are applied once, during the first database scan, while Strategy 3 is applied during LU-list construction. The proofs that these strategies are correct are ommitted due to space limitation. But they can be easily derived from the previous definitions.

4 Experimental Evaluation

Experiments were performed to assess the performance of LHUI-Miner on a computer having an Intel Xeon E3-1270 v5 processor running Windows 10, and 16 GB of free RAM. The performance of LHUI-Miner were compared with non-optimized versions and the HUI-Miner algorithm for mining HUIs. Four real-life datasets commonly used in the HUIM litterature were used: *mushroom*, *retail*, *kosarak* and *e-commerce*. 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. For the other datasets, external utilities 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 [4,6]. Besides, the timestamps of transactions in these three databases are generated

by adopting the same distribution as the e-commerce database. The source code of algorithms and datasets can be downloaded from the SPMF library [11].

LHUI-Miner was run with $minLength = 90$ *days* for *e-commerce* and 30 days for the other datasets. Thereafter, *lhui-op* denotes LHUI-Miner with optimizations; and *lhui-non-op* denotes LHUI-Miner without optimization. Algorithms were run on each dataset, while decreasing $lMinutil$ (for HUI-Miner $minutil = lMinutil \times \lceil \frac{W_D}{minLength} \rceil$) until they became too long to execute, ran out of memory or a clear trend was observed. Figure 1 compares the execution times of LHUI-Miner with and without optimization. Figure 2 compares the numbers of LHUIs and HUIs, respectively generated by these algorithms.

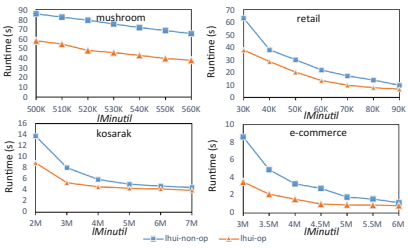

Fig. 1. Execution times

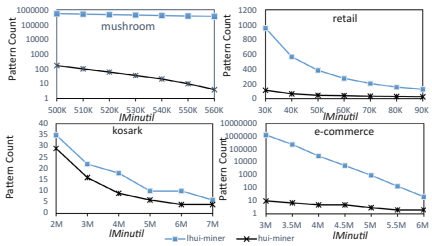

Fig. 2. Number of patterns found

It can be observed that in most cases, optimizations reduce the runtime. In some cases, the optimized algorithm is one time faster than the non-optimized algorithm, while in some cases there is little improvement. We also compared the execution time of HUI-Miner ($lMinutil \times \lceil \frac{W_D}{minLength} \rceil$) with the proposed algorithm. HUI-Miner is generally much faster because HUI-Miner generates much less patterns than LHUI-Miner (in other words, the problem of HUI mining is easier). However, when the number of patterns found by HUI-Miner is similar to the proposed algorithm, their runtimes are similar. Thus, because HUI-Miner is defined for a different problem, its results are not shown in Fig. 1.

A second observation is that the number of LHUIs 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 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 and 549,479 LHUIs. Among all patterns found, some interesting LHUIs are found in *e-commerce*. For instance, for $lMinutil = 1,432,360$ and $minlength = 90$ *days*, the itemset {pink polkadot bag, black and white baroque bag} has a PLHUI period from 2011/07/19 to 2011/11/02 where it generates a high utility, while the itemset is not a HUI in the whole database for $minutil = 6,000,000$. This information can be very useful for a retail store manager as it shows that this product generates a high profit from the end of July to early November. This can be useful to replenish stocks and offer promotions on this set of products during that period for the following year. Lastly, another observation is that for *kosarak*, the difference between the

number of LHUIs and HUIs is very small compared to other datasets. The reason is that the utilities of patterns do not vary much over time in *kosarak*.

5 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). An algorithm named LHUI-Miner (Local High-Utility Itemset Miner) was designed to efficiently discover LHUIs. Besides, three strategies were proposed to improve the performance of LHUI-Miner. An experimental evaluation has shown that LHUI-Miner can discover useful patterns that traditional HUIM could not find and that strategies reduces the runtime and memory consumption.

For future work, we will design concise representations of LHUIs to show a summary of all patterns to the user. Moreover, we will adapt the concept of this paper to other pattern mining problems such as sequential pattern mining and episode mining.

References

1. Agrawal, R., Srikant, R.: Fast algorithms for mining association rules in large databases. In: Proceedings of 20th International Conference on Very Large Databases, pp. 487–499. Morgan Kaufmann, Santiago de Chile (1994)
2. Zaki, M.J.: Scalable algorithms for association mining. IEEE Trans. Knowl. Data Eng. **12**(3), 372–390 (2000)
3. Fournier-Viger, P., Wu, C.-W., Zida, S., Tseng, V.S.: FHM: faster high-utility itemset mining using estimated utility co-occurrence pruning. In: Andreasen, T., Christiansen, H., Cubero, J.-C., Raś, Z.W. (eds.) ISMIS 2014. LNCS (LNAI), vol. 8502, pp. 83–92. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-08326-1_9
4. Liu, M., Qu, J.: Mining high utility itemsets without candidate generation. In: Proceedings of the 23rd ACM International Conference on Information and Knowledge Management, pp. 55–64. ACM, Maui (2012)
5. Liu, Y., Liao, W., Choudhary, A.: A two-phase algorithm for fast discovery of high utility itemsets. In: Ho, T.B., Cheung, D., Liu, H. (eds.) PAKDD 2005. LNCS (LNAI), vol. 3518, pp. 689–695. Springer, Heidelberg (2005). https://doi.org/10.1007/11430919_79
6. 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)
7. Wan, Q., An, A.: Discovering transitional patterns and their significant milestones in transaction databases. IEEE Trans. Knowl. Data Eng. **21**(12), 1692–1707 (2009)
8. 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. Adv. Eng. Inform. **33**, 29–43 (2017)
9. Loglisci, C., Ceci, M., Malerba, D.: Relational mining for discovering changes in evolving networks. Neurocomputing **150**, 265–288 (2015)

10. Fournier-Viger, P., Zida, S.: FOSHU: faster on-shelf high utility itemset mining-with or without negative unit profit. In: Proceedings of 30th Annual ACM Symposium on Applied Computing, pp. 857–864. ACM, Salamanca (2015)
11. Fournier-Viger, P., et al.: The SPMF open-source data mining library version 2. In: Berendt, B., et al. (eds.) ECML PKDD 2016. LNCS (LNAI), vol. 9853, pp. 36–40. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-46131-1_8