Discovering High Utility Change Points in Customer Transaction Data

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

[1] School of Natural Sciences and Humanities, Harbin Institute of Technology (Shenzhen), Shenzhen, China
philfv@hit.edu.cn

[2] School of Computer Sciences and Technology, Harbin Institute of Technology (Shenzhen), 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] Department of Computer Sciences, University of Auckland, Auckland, New Zealand
ykoh@cs.auckland.ac.nz

Abstract. High Utility Itemset Mining (HUIM) consists of identifying all sets of items (itemsets) that have a high utility (e.g. have a high profit) in a database of customer transactions. Important limitations of traditional HUIM algorithms is that they do not consider that the utility of itemsets varies as time passes and that itemsets may not have a high utility in the whole database, but in some specific time periods. To overcome these drawbacks, this paper defines the novel problem of discovering change points of high utility itemsets, that is to find the time points where the utility of an itemset is changing considerably. An efficient algorithms named HUCP-Miner is proposed to mine these change points. Experimental results show that the proposed algorithm has excellent performance and can discover interesting patterns that are not identified by traditional HUIM algorithms.

Keywords: Pattern mining · High-utility itemsets · Change points

1 Introduction

Frequent Itemset Mining (FIM) [1] is a well-studied task in data mining. Given a database of customer transactions, FIM consists of enumerating all sets of items (itemsets) that frequently appear together. *High-Utility Itemset Mining* (HUIM) [2,4–7] is a generalization of FIM. It consists of finding all itemsets having a utility (e.g. importance or profit) that is equal or greater than a user-defined threshold in a customer transaction database. HUIM is more challenging than FIM because the utility measure used in HUIM is not monotonic nor anti-monotonic, that is an itemset may have a utility that is smaller, equal or greater than that of its supersets or subsets. Because of this, search space reduction techniques of traditional FIM algorithms [1] cannot be used to reduce the search space of HUIM. To design HUIM algorithms that efficiently enumerate all high utility itemsets, upper-bounds on the utility measure have been proposed, which have anti-monotonic properties, to reduce the search space [5]. To mine high utility itemsets, several algorithms were designed such as UP-Growth [6], HUI-Miner [4] and FHM [2]. They use different search strategies and data structures.

HUIM has several applications such as biomedical applications, customer behavior analysis, and click stream analysis [4,6,7]. But a drawback of traditional HUIM is that the timestamps of transactions are not taken into account, although they provide valuable information that can be used to understand how the utility of itemsets changes over time. A few studies about FIM and HUIM have considered time. For instance, *periodic high-utility itemset mining* [8] aims at finding itemsets that periodically appear and yield a high profit in transactional data. Although this work consider the sequential ordering of transactions, it ignores timestamps. Moreover, the algorithm presented in that study tends to discover patterns having a utility that is stable over time. An algorithm was proposed for detecting time points where the frequency of itemsets changes in data streams [9]. Although it considers the sequential ordering of transactions, it ignores timestamps. Hence, it assumes that all consecutive transactions have the same time gap, which is unrealistic.

As time passes, the utility of itemsets may change. It is desirable to find the time points where the utility of itemsets changes dramatically. Discovering this information allow understanding trends in the data to support decision-making. For example, discovering that an upward/downward trend in the utility of some items is happening at a time point allows to improve inventory management by either increasing retail stocks or implementing strategies to reduce stocks.

To fulfill this needs, this paper defines the problem of identifying change points of high utility itemsets, that is time points indicating a considerable increase or decrease of utility. To be able to discover change points that represents significant trends in the utility distributions of itemsets, this paper adapts the concept of moving average crossover, which is used in time series analysis. A change point occurs for an itemset when short-term and long-term moving averages of its utility cross. However, it is impractical to mine all the change points of all the itemsets in a database, and it is also unnecessary since some itemsets seldomly appear and yield a low utility. The solution of this paper is to only find the change points of local high utility itemsets [11]. For example, this can be used to discover that an itemset $\{sunglasses, suncream, beach\_towel\}$ has a change point at the begining of the summer representing an increase and another change point at the end of the summer representing a decrease, and this itemset may not be a HUI in the whole database.

The rest of this paper is organized as follows. Section 2 presents preliminaries and defines the problem of identifying change points of HUIs. Section 3 describes the designed algorithm. Section 4 reports results from an experimental evaluation. Finally, Sect. 5 concludes the paper.

2 Preliminaries and Problem Statement

Consider a set of products (items) sold in a retail store, denoted as $I = \{i_1, i_2, \ldots, i_n\}$. A customer transaction T is a set of items $T \subseteq I$. A *transaction database* contains multiple customer transactions $D = \{T_1, T_2, \ldots, T_m\}$, In addition, each transaction T has a timestamp denoted as $t(T)$, which may not be unique. Besides, a positive number $p(i)$ called external utility is defined for each item $i \in I$ to indicate its relative importance. In the context of customer transaction analysis, $p(i)$ can be defined as the unit profit of i. In each transaction T, a positive number $q(i, T)$ called internal utility specifies the purchase quantity of each item i. For instance, a small transaction database is provided in Table 1. This database contains eight transactions, denoted as $T_1, T_2, \ldots, T_8$, and five items, denoted as a, b, c, d, e, where purchase quantities are shown as integers beside items. This database will be used as running example for the rest of this paper. In this database, timestamps of transactions $T_1, T_2 \ldots T_8$ are denoted as $d_1, d_3, \ldots d_{10}$. These values can represent days ($d_i = i$-th day) or other time units such as seconds, or milliseconds. Consider transaction T_2. It indicates that 4, 3, 2 and 1 units of items b, c, d and e were purchased, respectively. According to Table 2, which provides the external utility values of items, the unit profits of these items are 2, 1, 2 and 3, respectively.

Table 1. A transaction database

Transaction	Items $(item, purchase\_quantity)$	Time
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). Let there be an item i and a transaction T from a transaction database. The utility of i in T is defined and calculated as $u(i, T) = p(i) \times q(i, T)$. Let X be an *itemset* (a set of items), i.e. $X \subseteq I$. The utility of X in T is defined as $u(X, T) = \sum_{i \in X \wedge X \subseteq T} u(i, T)$. The utility of X in the database is defined as $u(X) = \sum_{T \in D \wedge X \subseteq T} u(X, T)$.

For instance, 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 called *high utility itemset* (HUI) if its utility $u(X)$ is no less than a user-specified positive threshold *minutil* [5]. HUIM is the task of finding all HUIs in a transaction database. The utility measure is not anti-monotonic, that is a superset of a HUI may not be a HUI. 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. $TWU(X) = \sum_{X \subseteq T \wedge T \in D} tu(T)$.

For 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 $TWU(X) < minutil$, all supersets of X can be pruned.

Limitations of HUIM are that transaction timestamps are ignored and that an itemset's utility may vary over time. To identify time periods where an itemset has a high utility, a concept of time window is used. It will then be used to identify change points in the utility of HUIs indicating upward/downward trends.

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 $W_{i,j} \subsetneqq W_{k,l}$. The utility of an itemset X in a window $W_{i,j}$ is defined as $u_{i,j}(X) = \sum_{T \in W_{i,j} \wedge X \subseteq T} u(X, T)$.

For example, $W_{d_1,d_3} = \{T_1, T_2, T_3\}$, $length(W_{d_1,d_3}) = 3 - 1 + 1 = 3$, and W_{d_1,d_3} subsumes W_{d_3,d_3}. But W_{d_2,d_3} does not subsume W_{d_3,d_3} because $W_{d_2,d_3} = \{T_2, T_3\} = W_{d_3,d_3}$. The utility of $\{b, c\}$ in the window W_{d_1,d_3} is $u_{d_1,d_3}(\{b, c\}) = u(\{b, c\}, T_1) + u(\{b, c\}, T_2) + u(\{b, c\}, T_3) = 6 + 11 + 6 = 23$.

A concept of change point is proposed by adapting the *moving average crossover* technique. In time series analysis, the *moving average* of a time-series is the mean of the previous n data points. It is used to smooth out short-term fluctuations. The larger n is, the more smoothing is obtained. The *moving average crossover* is the time points where two moving averages based on different degrees of smoothing cross each other. Moving average crossover is widely used by stock trading systems [10], where crossovers indicate points for buying/selling stocks. 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.

To find change points where an itemset's utility has upward/downward trends, this paper utilizes the concept of moving crossover. This allows to consider time points that are not equally spaced in time. The moving average utility of an itemset is defined as follows. Let there be a smoothing parameter γ representing a time length. The *moving average* utility of an itemset 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. For example, let $\gamma = 3$, the moving average utility of itemset $\{a,c\}$ at timestamp d_6 is $mau_3(\{a,c\},d_6) = \frac{u_{d_6-\frac{3-1}{2},d_6+\frac{3-1}{2}}(\{a,c\})}{3} = \frac{u_{d_5,d_7}(\{a,c\})}{3} = 28$.

To apply the concept of moving average crossover, it is necessary to define two moving averages, respectively having a short and a long window. For an itemset X and a timestamp t, the short term moving average utility $mau_{\gamma}(X,t)$ is calculated over the window $W_{t-\frac{\gamma-1}{2},t+\frac{\gamma-1}{2}}$. This window has a length of γ. The long term moving average utility $mau_{\lambda\times\gamma}(X,t)$ is calculated using a window $W_{t-\frac{\lambda\times\gamma-1}{2},t+\frac{\lambda\times\gamma-1}{2}}$ having length of $\lambda \times \gamma$, where $\lambda \geq 1$ is a user-defined parameter called the *moving average crossover coefficient*. Based on these windows, the concept *change point* is proposed, indicating time points where the utility of an itemset follows an upward or downward trend.

Definition 5 (Change point). For an itemset X, a time point t is a change point if $mau_{winLength}(X,t) \geq (\leq) mau_{\lambda\times winLength}(X,t) \wedge mau_{winLength}(X,t-1) < (>) mau_{\lambda\times winLength}(X,t-1)$, where $winLength$ and λ are two user-defined parameters ($winLength > 1$, $\lambda \geq 1$), which are the short term window length and the scaling factor to obtain the long term window. If $\lambda = 1$, the short and long moving average utility are equal and there is no change point. If $\lambda = \infty$, the change points of X are the time points where its short moving average utility crosses the average utility in the whole database.

For example, if $winLength = 3$ and $\lambda = 1.67$, d_5 is a change points of itemset $\{a,c\}$ because $mau_3(\{a,c\},d_5-1) = u_{d_3,d_5}(\{a,c\})/3 = 4 < mau_{3\times1.67}(\{a,c\},d_5-1) = u_{d_2,d_6}(\{a,c\})/5 = 5.6$, and $mau_3(\{a,c\},d_5) = 7 > mau_{3\times1.67}(\{a,c\},d_5-1) = 5.6$. Similarly, we can find that d_7 is also a change point of {a,c}. And it can be seen that d_5 is where itemset {a,c} begins to appear and d_7 is where itemset {a,c} begins to disappear in the database.

It is not necessary to find change points of every itemsets, since some seldomly appear or generate little utility. Instead, we only find change points of local high utility itemset [11], a concept defined as follows.

Definition 6 (Local High Utility Itemset). An itemset X is said to be a Local High Utility Itemset (LHUI) if there exists a time point t such that $mau_{winLength}(X,t) > minMau$, where $minMau$ and $winLength$ are two user-defined parameters [11].

Definition 7 (Mining High utility Change Points). The problem of Mining High utility Change Points is to find the change points of all LHUIs, given the parameters $winLength$, $minMau$ and λ, set by the user.

For example, given the database of Table 1, $winLength = 3$, $minMau = 10$ and $lambda = 1.67$, 24 Local High Utility Itemsets with their change points are found, including $\{d\}$:$\{d_3,d_6\}$, $\{b,d\}$:$\{d_3,d_6\}$ and $\{a,c\}$:$\{d_5,d_7,d_9,d_{10}\}$.

3 Proposed Algorithm

This paper proposes an algorithm to efficiently mine change points of LHUIs, named HUCP-Miner. It extends the LHUI-Miner [11] algorithm for LHUI mining. The HUCP-Miner algorithm explores 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 $Y = X \cup \{j\} \wedge \forall i \in X, j \succ i$. HUCP-Miner relies on a data structure called *Local Utility-list* (LU-list) [11] to store information about each itemset. This structure extends the utility-list [4] structure by storing information about time periods. HUCP-Miner initially scans the database to build a LU-list for each item. Then, it performs a depth-first search to explore the search space, while joining pairs of itemsets to generate their extensions and corresponding LU-lists. Checking if an itemset is a LHUI and calculating its moving average utility is done using is LU-list (without scanning the database). The LU-list structure is defined as follows. Let there be an itemset X. Its *LU-list* contains a tuple for each transaction where X appears. A *tuple* 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 maximum *LHUI periods* and *PLHUI periods* of X, respectively. The concepts of LHUI periods and PLHUI periods of X were defined in the LHUI-Miner algorithm [11].

A LU-list allow to derive key information about an itemset. First, if $iutilPeriods$ of an itemset X is not empty, then there exists at least a time point t, where $mau_{winLenth}(X, t) > minMau$. In other words, X is a LHUI. On the other hand, if $utilPeriods$ is empty, all transitive extensions of X are not LHUIs and can be pruned from the search space. The proof of this property is not presented due to the page limitation. The LU-list of an itemset also allows to calculate its moving average utility without scanning the database.

The HUCP-Miner Algorithm. The input of HUCP-Miner is a transaction database, $minMau$, $winLength$ and λ. HUCP-Miner outputs all LHUIs with their change points. The algorithm first reads the database once to obtain each item's TWU, and construct an array, called *tid2time*. The i-th position of *tid2time* contains the timestamp of transaction $t(T_i)$. Thereafter, HUCP-Miner only considers items having a TWU no less than $minMau \times winLength$, denoted as I^*. Based on the calculated TWU values of items, the total order $\succ$ on I^* is then set as the ascending order of TWU values [2]. The database is then read again and items in each transaction are reordered according to $\succ$ to create the LU-list of each item $i \in I^*$. Then, HUCP-Miner perform a depth-first search of itemsets by calling the recursive $HUCP\text{-}Search$ procedure with $\emptyset$, the LU-lists of 1-itemsets, $minMau$, $winLength$ and λ.

The input of the $HUCP\text{-}Search$ (Algorithm 1) procedure is (1) an itemset P, (2) extensions of P, (3) $minMau$, (4) $winLength$, and (5) λ. $HUCP\text{-}Search$ first verifies if $iutilPeriods$ is empty for each extension Px of P according to their respective LU-lists. If empty, then Px is a LHUI and it is output with its change points. Besides, if *utilPeriods* is not empty, extensions of Px will be explored. This is done by combining 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*, *utilPeriods* and change points of Pxy are constructed by calling the *generateCP* procedure. Then, *HUCP-Search* is called with Pxy to calculate its utility and explore its extension(s) using a depth-first search. The *HUCP-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 and their change points.

Algorithm 1. HUCP-Search

```
input : $P$: an itemset, $ExtensionsOfP$: extensions of $P$, $minMau$: a user-specified
        threshold, $winLength$: a window length threshold, $\lambda$: a user-specified parameter
output: the set of LHUIs and their change points
1  foreach itemset $Px \in ExtensionsOfP$ do
2      if $Px.LUList.iutilPeriods \neq \emptyset$ then output $Px$ with $Px.LUList.iutilPeriods$;
3      if $Px.LUList.utilPeriods \neq \emptyset$ then
4          $ExtensionsOfPx \leftarrow \emptyset$;
5          foreach itemset $Py \in ExtensionsOfP$ such that $y \succ x$ do
6              $Pxy.LUList \leftarrow$ Construct $(P, Px, Py)$;
7              generateCP $(Pxy, minMau, winLength)$;
8              $ExtensionsOfPx \leftarrow ExtensionsOfPx \cup Pxy$;
9          end
10         HUCP-Miner $(Px, ExtensionsOfPx, minutil, winLength, \lambda)$;
11     end
12 end
```

The *generateCP* procedure (Algorithm 2) takes as input (1) a LU-list lUl, (2) $minMau$, (3) $winLength$, and λ. 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 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 $winLength$, 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 $minMau$ 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, (4) the moving average utility of the center of a window can be calculated as $mau_\gamma = iutils/winLength$. Similarly, the long term moving average utility can be calculated using a larger window. Then, the moving average are compared to determine if the current window center is a change point. If so it is stored.

Algorithm 2. The generatePeriods procedure

```
input : $lUl$: a LU-list, $minMau$: a user-specified utility threshold, $winLength$: a
        user-specified window length threshold, $\lambda$: a user-specified parameter
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 $ul.get(winEnd).time \leq ul.get(winStart).time + winLength$ 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
       $iutils \geq minMau \times winLength$. Otherwise, add it to $lul.iutilPeriods$;
15     merge the $[winStart, winEnd]$ period with the previous period if
       $iutils + rutils \geq minMau \times winLength$. Otherwise, add it to $lul.utilPeriods$;
16     $mau_\gamma = iutils/winLength$, use similar strategy to calculate $mau_{\gamma\times\lambda}$, compare to
       determine if current center of window is a change point, if so, store it.
17 end
```

Optimizations. As HUCP-Miner is based on LHUI-Miner [11], all its optimizations can be used to improve the performance of HUCP-Miner.

4 Experimental Evaluation

An experiment was done to compare the performance of HUCP-Miner with a non-optimized version and the HUI-Miner algorithm for mining HUIs. Three datasets were used, which are commonly utilized to benchmark HUIM algorithms: *retail*, *kosarak* and *e-commerce*, representing the main types of data (long transactions, dense and sparse). Let $|I|$, $|D|$ and A represents the number of distinct items, transactions and average transaction length. *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 *retail* and *kosarak*, external utilities of items were 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]. Moreover, transactions timestamps for the three databases were generated using the same distribution as the e-commerce database. The source code of algorithms and datasets can be downloaded as part of the SPMF [3] open source data mining library at http://www.philippe-fournier-viger.com/spmf/.

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

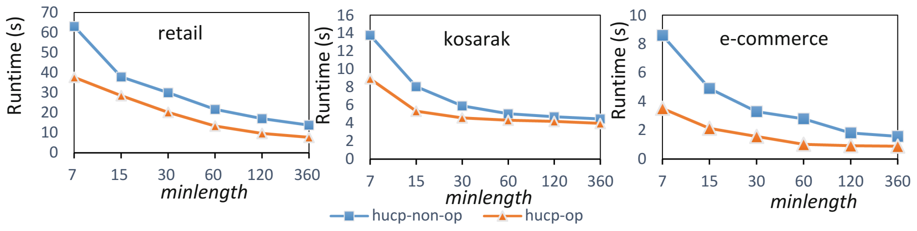

Fig. 1. Execution times

It can be observed that optimizations generally reduce the runtime. In some cases, HUCP-Miner can be one time faster than the non-optimized version. We also compared the execution time of HUI-Miner ($minutil = minMau \times \lceil \frac{W_D}{winLength} \rceil \times W_D$) with these algorithms. HUI-Miner is often much faster and produces much less patterns. However, when the number of patterns found by HUI-Miner is similar to the proposed algorithms, their runtimes are similar. Thus, because HUI-Miner is defined for a different problem its results are not shown in Fig. 1.

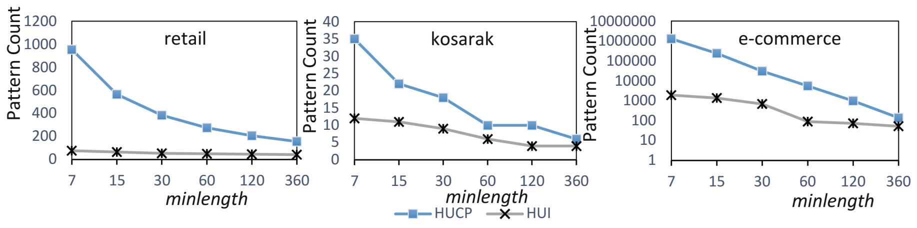

Fig. 2. Number of patterns found

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*), $minutil = 500,000$, $minMau = 83,333$, $winLength = 30$ *days*, there are 168 HUIs and 549,479 LHUIs. 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*.

Among all patterns found, some interesting high utility change points are found in *e-commerce*. For instance, for $minMau = 16,000$ and $winLength = 90$ *days*, the itemset {white hanging heart T-light holder} has a positive trend on 2011/3/6 and 2011/9/1, and a negative trend on 2011/6/13 and 2011/12/8, while the itemset is not a HUI in the whole database for $minutil = minMau \times 373 = 5,968,000$. This information is important for retail store management since it indicates that this item has a sale increase and decrease at the end of July and in early November, respectively. This can be useful to replenish stocks and offer promotions on this set of products at these time points for the following year.

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 high utility change points. Those are time points where an upward or downward trend is observed in the utility of local high utility itemsets. An experimental evaluation has shown that the proposed HUCP-Miner algorithm can discover useful patterns that traditional HUIM could not find and that optimizations reduced the runtime and memory consumption. For future work, we will adapt the concept of change points developed in 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 the 20th International Conference on Very Large Databases, pp. 487–499. Morgan Kaufmann, Santiago (1994)
2. 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
3. Fournier-Viger, P., Gomariz, A., Gueniche, T., Soltani, A., Wu, C., Tseng, V.S.: SPMF: a Java open-source pattern mining library. J. Mach. Learn. Res. (JMLR) **15**, 3389–3393 (2014)
4. Liu, M., Qu, J.: Mining high utility itemsets without candidate generation. In: 22nd 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. Peng, A.Y., Koh, Y.S., Riddle, P.: mHUIMiner: a fast high utility itemset mining algorithm for sparse datasets. In: Proceedings of 22nd Pacific-Asia Conference on Knowledge Discovery and Data Mining, pp. 196–207. ACM (2017)
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. Wan, Q., An, A.: Discovering transitional patterns and their significant milestones in transaction databases. IEEE Trans. Knowl. Data Eng. **21**(12), 1692–1707 (2009)
10. Ni, Y., Liao, Y.C., Huang, P.: MA trading rules, herding behaviors, and stock market overreaction. Int. Rev. Econ. Finan. **39**, 253–265 (2015)
11. Fournier-Viger, P., Zhang, Y., Lin, J.C.-W., Fujita, H., Koh, Y.S.: Mining local high utility itemsets. In: Hartmann, S., Ma, H., Hameurlain, A., Pernul, G., Wagner, R.R. (eds.) DEXA 2018. LNCS, vol. 11030, pp. 450–460. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-98812-2_41