FHN: Efficient Mining of High-Utility Itemsets with Negative Unit Profits

Philippe Fournier-Viger

Dept. of Computer Science, University of Moncton, Canada
philippe.fournier-viger@umoncton.ca

Abstract. High utility itemset (HUI) mining is a popular data mining task. It consists of discovering sets of items generating high profit in a transaction database. Several efficient algorithms have been proposed for this task. But few can handle items with negative unit profits despite that such items occurs in many real-life transaction databases. Mining HUIs in a database where items have positive and negative unit profits is a very computationally expensive task. To address this issue, we present an efficient algorithm named FHN (Faster High-Utility itemset miner with Negative unit profits). FHN discovers HUIs without generating candidates and introduces several strategies to handle items with negative unit profits efficiently. Experimental results with six real-life datasets shows that FHN is up to 500 times faster and can use up to 250 times less memory than the state-of-the-art algorithm HUINIV-Mine.

Keywords: frequent pattern mining, high-utility itemset mining, negative unit profit, negative weights, transaction database

1 Introduction

Frequent Itemset Mining (FIM) [2] is a popular data mining task that is essential to a wide range of applications. Given a transaction database, FIM consists of discovering frequent itemsets. i.e. groups of items (itemsets) appearing frequently in transactions [2]. However, an important limitation of FIM is that it assumes that each item cannot appear more than once in each transaction and that all items have the same importance (weight, unit profit or value). These assumptions often do not hold in real applications. For example, consider a database of customer transactions containing information about the quantities of items in each transaction and the unit profit of each item. FIM algorithms would discard this information and may thus discover many frequent itemsets generating a low profit and fail to discover less frequent itemsets that generate a high profit. To address this issue, the problem of FIM has been redefined as *High-Utility Itemset Mining* (HUIM) to consider the case where items can appear more than once in each transaction and where each item has a weight (e.g. unit profit). The goal of HUIM is to discover high utility itemsets (HUIs), i.e. itemsets generating a high profit. HUIM has a wide range of applications such as website click stream analysis, cross-marketing in retail stores and biomedical applications [3, 9, 12]. HUIM

has also inspired several important data mining tasks such as high-utility sequential pattern mining [15, 16], high-utility episode mining [14] and high-utility stream mining [11].

The problem of HUIM is widely recognized as more difficult than the problem of FIM. In FIM, the *downward-closure property* states that the support of an itemset is anti-monotonic, that is the supersets of an infrequent itemset are infrequent and subsets of a frequent itemset are frequent. This property is very powerful to prune the search space. In HUIM, the utility of an itemset is neither monotonic or anti-monotonic, that is a high utility itemset may have a superset or subset with lower, equal or higher utility [2]. Thus, techniques to prune the search space developed in FIM cannot be directly applied in HUIM.

Many studies have been carried to develop efficient HUIM algorithms [3, 8, 4, 9, 10, 12, 13]. However, these algorithms are not designed to handle items having negative weights/unit profits, despite that such items occur in many real-life transaction databases. For example, it is common that retail stores sell items at a loss (e.g. printers) to stimulate the sale of other related items (e.g. proprietary printer cartridges). It was demonstrated that if classical HUIM algorithms are applied on databases containing items with negative unit profits, they can generate an incomplete set of HUIs [1]. The reason is that these algorithms over-estimate the utility of itemsets to prune the search space. But, when items with negative unit profits are considered, these estimations may become underestimations, and thus HUIs may be pruned. The state-of-the-art algorithm for mining HUIs while considering negative unit profits is HUINIV-Mine [1]. However, mining HUIs with negative unit profits remains very costly in terms of execution time and memory. Therefore, an important challenge is to design a more efficient algorithm for this task.

In this paper, we address this challenge. We present a novel algorithm named FHN (Fast High-utility itemset miner with Negative unit profits) to mine HUIs while considering both positive and negative unit profits. It extends the current fastest HUI mining algorithm named FHM [4] so that it can handle negative unit profits efficiently. We compare the performance of FHN and HUINIV-Mine on six real-life datasets. Results show that FHN is up to 500 times faster than HUINIV-Mine and consumes up to 250 times less memory. The rest of this paper is organized as follows. Section 2, 3, 4 and 5 respectively presents the problem definition and related work, the FHN algorithm, the experimental evaluation and the conclusion.

2 Problem definition and related work

We first introduce important preliminary definitions.

Definition 1 (transaction database). Let I be a set of items (symbols). A *transaction database* is a set of transactions $D = \{T_1, T_2, ..., T_n\}$ such that for each transaction T_c, $T_c \in I$ and T_c has a unique identifier c called its Tid. Each item $i \in I$ is associated with a positive or negative number $p(i)$, called

its external utility (e.g. unit profit). For each transaction T_c such that $i \in T_c$, a positive number $q(i, T_c)$ is called the internal utility of i (e.g. purchase quantity).

Example 1. Consider the database of Fig. 1 (left), which will be used as our running example. This database contains five transactions $(T_1, T_2 ... T_5)$. Transaction T_2 indicates that items a, c, e and g appear in this transaction with an internal utility of respectively 2, 6, 2 and 5. Fig. 1 (right) indicates that the external utility of these items are respectively -5, 1, 3 and 1. Thus, item a is sold at loss.

Definition 2 (utility of an item/itemset in a transaction). The utility of an item i in a transaction T_c is denoted as $u(i, T_c)$ and defined as $p(i) \times q(i, T_c)$. The utility of an itemset X (a group of items $X \subseteq I$) in a transaction T_c is denoted as $u(X, T_c)$ and defined as $u(X, T_c) = \sum_{i \in X} u(i, T_c)$.

TID	Transactions
T_1	(a,1)(c,1)(d,1)
T_2	(a,2)(c,6)(e,2)(g,5)
T_3	(a,1)(b,2)(c,1)(d,6),(e,1),(f,5)
T_4	(b,4)(c,3)(d,3)(e,1)
T_5	(b,2)(c,2)(e,1)(g,2)

Item	a	b	c	d	e	f	g
Profit	-5	2	1	2	3	1	1

Fig. 1. A transaction database (left) and external utility values (right)

Example 2. The utility of item e in T_2 is $u(e, T_2) = 3 \times 2 = 6$. The utility of the itemset $\{c, e\}$ in T_2 is $u(\{c, e\}, T_2) = u(c, T_2) + u(e, T_2) = 1 \times 6 + 3 \times 2 = 12$.

Definition 3 (utility of an itemset in a database). The utility of an itemset X is denoted as $u(X)$ and defined as $u(X) = \sum_{T_c \in g(X)} u(X, T_c)$, where $g(X)$ is the set of transactions containing X.

Example 3. The utility of the itemset $\{c, e\}$ is $u(\{c, e\}) = (u(c, T_2) + u(e, T_2)) + (u(c, T_3) + u(e, T_3)) + (u(c, T_4) + u(e, T_4)) + (u(c, T_5) + u(e, T_5)) = (6 + 6) + (1 + 3) + (3 + 3) + (2 + 3) = 27$. The utility of the itemset $\{a, d, f\}$ is $u(\{a, d, f\}) = (u(a, T_3) + u(d, T_3)) + u(f, T_3)) = -5 + 12 + 5 = 12$.

Definition 4 (problem of HUI mining with/without negative unit profits). Let *minutil* be a threshold set by the user. An itemset X is a *high-utility itemset* if $u(X) \geq minutil$. Otherwise, X is a *low-utility itemset*. The *problem of high-utility itemset mining* is to discover all high-utility itemsets in a database where external utility values are positive. The *problem of high-utility itemset mining with negative unit profits* is to discover all high-utility itemsets in a database where external utility values may be positive or negative.

Example 4. If $minutil = 20$, twenty HUIs should be found in the database. They are $\{a, b, c, d, e, f\}$:20, $\{b, d, f\}$:21, $\{b, d, e, f\}$:24, $\{b, c, d, e, f\}$:25, $\{b, c, d, f\}$:22,

$\{d, e, f\}$:20, $\{c, d, e, f\}$:21, $\{c, e, g\}$:24, $\{d\}$:20, $\{b, d\}$:30, $\{b, d, e\}$:36, $\{b, c, d, e\}$:40, $\{b, c, d\}$:34, $\{d, e\}$:24, $\{c, d, e\}$:28, $\{c, d\}$:25, $\{b, e\}$:25, $\{b, c, e\}$:31, $\{b, c\}$:22 and $\{c, e\}$:27, where the number following each itemset is its utility.

Two known properties of HUIs with respect to items having negative unit profits are the following.

Property 1 (a HUI may contain items having negative external utilities). It can be clearly seen from the example that a HUI may contain items having a negative external utility value. For example, $\{a, b, c, d, e, f\}$ contains a, which has an external utility of -5.

Property 2 (a HUI must contain at least an item having a positive external utility [1]). Although a HUI may or may not contain items having negative external utility values, a HUI need to contain at least an item having a positive external utility value (otherwise its utility would be negative and it would not be a HUI).

It can be demonstrated that the utility measure is not monotonic or anti-monotonic. In other words, an itemset may have a utility lower, equal or higher than the utility of its subsets. Therefore, the strategies that are used in FIM to prune the search space based on the anti-monotonicity of the support cannot be directly applied to discover high-utility itemsets. Several HUIM algorithms circumvent this problem by overestimating the utility of itemsets using a measure called the Transaction-Weighted Utilization (TWU) [3, 10, 12], which is anti-monotonic. The TWU measure assumes that all items have positive external utility values. The TWU measure is defined as follows.

Definition 5 (transaction utility). The *transaction utility* (TU) of a transaction T_c is the sum of the utility of the items from T_c in T_c. i.e. $TU(T_c) = \sum_{x \in T_c} u(x, T_c)$.

Definition 6 (transaction weighted utilization). The *transaction-weighted utilization* (TWU) of an itemset X is defined as the sum of the transaction utility of transactions containing X, i.e. $TWU(X) = \sum_{T_c \in g(X)} TU(T_c)$.

Example 5. Consider the database of the running example and that the external utility value of item a is 5 rather than -5 ($p(a) = 5$). The TU of transactions T_1, T_2, T_3, T_4 and T_5 are respectively 8, 27, 30, 20 and 11. The TWU of items a, b, c, d, e, f and g are 65, 61, 96, 58, 88, 30 and 38. Consider item b. $TWU(\{b\}) = TU(T_3) + TU(T_4) + TU(T_5) = 30 + 20 + 11 = 61$.

The TWU measure has three important properties that are used to prune the search space. These properties only hold if external utility values of items are positive [1].

Property 3 (overestimation). The TWU of an itemset X is higher than or equal to its utility, i.e. $TWU(X) \geq u(X)$ [10].

Property 4 (antimonotonicity). The TWU measure is anti-monotonic. Let X and Y be two itemsets. If $X \subset Y$, then $TWU(X) \geq TWU(Y)$ [10].

Property 5 (pruning). Let X be an itemset. If $TWU(X) < minutil$, then the itemset X is a low-utility itemset as well as all its supersets [10].

Most algorithms for high utility mining that only handle positive external utility values (e.g. Two-Phase [10], IHUP [3] and UPGrowth [12]) utilizes Property 5 to prune the search space. They operate in two phases. In Phase 1, they identify candidate high-utility itemsets by calculating their TWU. In Phase 2, they scan the database to calculate the exact utility of all candidates found in Phase 1 to eliminate low-utility itemsets. However, if such algorithms are applied on databases containing negative unit profits, some HUIs may not be output. For example, consider the running example. If item a has an external utility of -5 rather than 5 as we previously considered, $TWU(\{c, e, g\}) = TWU(T_2) = 7$ and $u(\{c, e, g\}) = 17$, which would be a violation of Property 5 stating that the TWU of an itemset is an overestimation of its utility. The consequence is that if $minutil = 10$, the itemset $\{c, e, g\}$ would not be output by algorithms relying on that property even though this itemset is a HUI (it would be pruned because $TWU(\{c, e, g\}) < minutil$).

To mine high utility itemsets while considering both positive and negative unit profits and output the full set of HUIs, the state-of-the-art algorithm is HUINIV-Mine [1]. It is an extension of the Two-Phase algorithm [10]. To avoid the aforementioned problem, HUINIV-Mine redefines the notion of transaction utility as follows (and thus the TWU measure).

Definition 7 (redefined transaction utility). The *redefined transaction utility* of a transaction T_c is the sum of the utility of the items from T_c having positive external utilities. i.e. $TU(T_c) = \sum_{x \in T_c \wedge p(x) > 0} u(x, T_c)$.

Example 6. Fig. 2 (left) shows the redefined TU of transactions T_1, T_2, T_3, T_4, T_5 for the running example. Fig. 2 (right) shows the TWU of single items based on the redefined transaction utility values. Consider itemsets $\{c, e, g\}$ and $\{e, g\}$. The values $TWU(\{c, e, g\})$ and $TWU(\{e, g\})$ are equal to 17, which are overestimations of $u(\{c, e, g\}) = 17$ and $u(\{e, g\}) = 11$.

Using the redefined transaction utility restores Property 5. This is what allows HUINIV-Mine to find the complete set of HUIs. However, a major problem is that the task of mining HUIs while considering both positive and negative unit profits remain computationally very expensive both in terms of execution time and memory, especially for datasets containing dense or long transactions. It is thus a challenge to build more efficient algorithms.

To address this issue, in this paper, we propose an algorithm named FHN that is a variation of the FHM algorithm [4]. FHM is a recently proposed algorithm for HUI mining, which is designed to handle only positive external utility values. FHM provides the benefit of mining HUIs in a single phase, thus avoiding the candidate generation step of other HUI mining algorithms such as Two-Phase, UPGrowth and IHUP. FHM utilizes the depth-first search procedure and

utility-list structure recently introduced in HUI-Miner [9] to explore the search space of itemsets, but also provides an efficient optimization named EUCP (Estimated Utility Co-occurrence Pruning) that makes FHM up to 6 times faster than HUI-Miner. FHM associates a *utility-list* [9] to each pattern. Utility-lists allow calculating the utility of a pattern quickly by making join operations with utility-lists of smaller patterns. Utility-lists are defined as follows.

Definition 8 (Utility-list). Let $\succ$ be any total order on items from I. The *utility-list* of an itemset X in a database D is a set of tuples such that there is a tuple $(tid, iutil, rutil)$ for each transaction T_{tid} containing X. The *iutil* element of a tuple is the utility of X in T_{tid}. i.e., $u(X, T_{tid})$. The *rutil* element of a tuple is defined as $\sum_{i \in T_{tid} \wedge i \succ x \forall x \in X} u(i, T_{tid})$.

Example 7. Assume that $\succ$ is the alphabetical order and that the external utility of item a is 5 rather than -5. The utility-list of $\{a\}$ is $\{(T_1, 5, 3), (T_2, 10, 17), (T_3, 5, 25)\}$. The utility-list of $\{d\}$ is $\{(T_1, 2, 0), (T_3, 12, 8), (T_4, 6, 3)\}$. The utility-list of $\{a, d\}$ is $\{(T_1, 7, 0), (T_3, 17, 8)\}$.

FHM discovers HUIs by performing a single database scan to create utility-lists of patterns containing single items. Then, longer patterns are obtained by performing the join operation of utility-lists of shorter patterns. The join operation for single items is performed as follows. Consider two items x, y such that $x \succ y$, and their utility-lists $ul(\{x\})$ and $ul(\{y\})$. The utility-list of $\{x, y\}$ is obtained by creating a tuple $(ex.tid, ex.iutil + ey.iutil, ey.rutil)$ for each pairs of tuples $ex \in ul(\{x\})$ and $ey \in ul(\{y\})$ such that $ex.tid = ey.tid$. The join operation for two itemsets $P \cup \{x\}$ and $P \cup \{y\}$ such that $x \succ y$ is performed as follows. Let $ul(P)$, $ul(\{x\})$ and $ul(\{y\})$ be the utility-lists of P, $\{x\}$ and $\{y\}$. The utility-list of $P \cup \{x, y\}$ is obtained by creating a tuple $(ex.tid, ex.iutil + ey.iutil - ep.iutil, ey.rutil)$ for each set of tuples $ex \in ul(\{x\})$, $ey \in ul(\{y\})$, $ep \in ul(P)$ such that $ex.tid = ey.tid = ep.tid$. Calculating the utility of an itemset using its utility-list and pruning the search space is done as follows.

Property 6 (Calculating utility of an itemset using its utility-list). The utility of an itemset is the sum of *iutil* values in its utility-list [9].

Property 7 (Pruning search space using utility-lists). Let X be an itemset. Let the *extensions* of X be the itemsets that can be obtained by appending an item y to X such that $y \succ i$, $\forall i \in X$. If the sum of *iutil* and *rutil* values in $ul(X)$ is less than *minutil*, then X and its extensions are low utility [9].

Before presenting our algorithm, we demonstrate with an example that the pruning property used by FHM and HUI-Miner is invalid if both negative and positive external utility values appears in a database. Consider that item a and d in the running example have respectively external utility values of 5 and -2, and that $minutil = 10$. The utility-list of $\{a, b\}$ would thus contains a single element, which is $\{(T_3, 9, -3)$. According to Property 7, because the sum of *iutil* and *rutil* values in $ul(\{a, b\})$ is 6, which is less than *minutil*, $\{a, b\}$ and its extensions are low utility. However, this is not the case since itemset $u(\{a, b, f\}) = 14$. Because of this, FHM and HUI-Miner would not find this HUI.

TID	TU
T_1	3
T_2	17
T_3	25
T_4	20
T_5	11

Item	TWU
a	45
b	56
c	77
d	48
e	74
f	25
g	28

Item	a	b	c	d	e	f
b	25					
c	45	56				
d	28	45	48			
e	42	56	74	45		
f	25	25	25	25	25	
g	17	11	28	0	28	0

Fig. 2. Transaction utilities (left), TWU values of single items (center) and EUCS (right)

3 The FHN algorithm

In this section, we present our proposal, the FHN algorithm. We first describe the main procedure, which is inspired by the FHM [4] algorithm. This procedure can only handle positive external utility values. We then explain how it is adapted to handle negative unit profits without missing any HUIs. We call this new algorithm the FHN algorithm.

3.1 Main procedure

The main procedure (Algorithm 1) takes as input a transaction database with utility values and the *minutil* threshold. The algorithm first scans the database to calculate the TWU of each item. Then, the algorithm identifies the set I^* of all items having a TWU no less than *minutil* (other items are ignored since they cannot be part of a high-utility itemset by Property 3). The TWU values of items are then used to establish a total order $\succ$ on items, which is the order of ascending TWU values (as suggested in [9]). A second database scan is then performed. During this database scan, items in transactions are reordered according to the total order $\succ$, the utility-list of each item $i \in I^*$ is built and a structure named EUCS (Estimated Utility Co-Occurrence Structure) is built [4]. This latter structure stores the TWU of all pairs of items $\{a, b\}$ such that $u(\{a, b\}) \neq 0$. As suggested in FHM, the EUCS is implemented as a hashmap of hashmaps since in practice a limited number of pairs of items co-occurs in transactions (see [4] for more details). Building the EUCS is very fast (it is performed with a single database scan) and occupies a small amount of memory, bounded by $|I^*| \times |I^*|$, although in practice the size is much smaller because a limited number of pairs of items co-occurs in transactions. After the construction of the EUCS, the depth-first search exploration of itemsets starts by calling the recursive procedure *Search* with the empty itemset $\emptyset$, the set of single items I^*, *minutil* and the EUCS.

The *Search* procedure (Algorithm 2) takes as input (1) an itemset P, (2) extensions of P having the form Pz meaning that Pz was previously obtained by appending an item z to P, (3) *minutil* and (4) the EUCS. The search procedure operates as follows. For each extension Px of P, if the sum of the *iutil* values

Algorithm 1: The FHN algorithm

input : D: a transaction database, *minutil*: a user-specified threshold
output: the set of high-utility itemsets

1 Scan D to calculate the TWU of single items;
2 $I^* \leftarrow$ each item i such that $TWU(i) \geq minutil$;
3 Let $\succ$ be the total order of TWU ascending values on I^*;
4 Scan D to built the utility-list of each item $i \in I^*$ and build the $EUCS$ structure;
5 `Search` ($\emptyset$, I^*, *minutil*, $EUCS$);

of the utility-list of Px is no less than *minutil*, then Px is a high-utility itemset and it is output (cf. Property 4). Then, if the sum of *iutil* and *rutil* values in the utility-list of Px are no less than *minutil*, it means that extensions of Px should be explored (cf. Property 7). This is performed by merging Px with all extensions Py of P such that $y \succ x$ and $TWU(\{x, y\}) \geq minutil$, to form extensions of the form Pxy containing $|Px| + 1$ items. The utility-list of Pxy is then constructed as in FHM by calling the *Construct* procedure (cf. Algorithm 3) to join the utility-lists of P, Px and Py. This latter procedure is the same as in FHM [4] and is thus not detailed here. Then, a recursive call to the *Search* procedure with Pxy is done to calculate its utility and explore its extension(s). Since the *Search* 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 Property 7. It can be easily seen based on Properties 6 and 7 that this procedure is correct and complete to discover all high-utility itemsets.

3.2 Modifying the algorithm to handle negative item unit profits

We next explain how the algorithm is modified to handle negative unit profits. Let the term "positive items" and "negative items" denote items respectively having positive and negative external utility values. To be able to transform the algorithm described in the previous subsection into an algorithm that outputs all HUIs when both negative and positive items are used, we first make a few novel and very important observations that were not done or used in HUINIV-Mine.

First, we define the total order $\succ$ such that negative items always succeed all positive items. Now consider an itemset X. Let $up(X) \subseteq X$ be the set of all positive items in X. Moreover, let $un(X) \subseteq X$ be the set of all negative items in X. We have the following important properties.

Property 8 (upper bound on utility using positive items). Let X be an itemset. It follows that $u(X) \leq u(up(X))$. **Rationale.** This property holds because $X \setminus up(X) = un(X)$ and negative items can only decrease the utility of X.

Property 9 (downward closure of extensions with negative items). Let X be an itemset and z be a negative item such that $z \notin X$. It follows that $u(up(X \cup \{z\})) \leq u(up(X))$. **Rationale.** Clearly, $up(X) = up(X \cup \{z\})$. Moreover, the

Algorithm 2: The *Search* procedure

input : P: an itemset, *ExtensionsOfP*: a set of extensions of P, the *minutil* threshold, the $EUCS$ structure
output: the set of high-utility itemsets

```
1  foreach itemset Px ∈ ExtensionsOfP do
2      if SUM(Px.utilitylist.iutils) ≥ minutil then
3          output Px;
4      end
5      if SUM(Px.utilitylist.iutils)+SUM(Px.utilitylist.rutils) ≥ minutil then
6          ExtensionsOfPx ← ∅;
7          foreach itemset Py ∈ ExtensionsOfP such that y ≻ x do
8              if TWU({x, y}) ≥ minutil) then
9                  Pxy ← Px ∪ Py;
10                 Pxy.utilitylist ← Construct (P, Px, Py);
11                 ExtensionsOfPx ← ExtensionsOfPx ∪ Pxy;
12             end
13         end
14         Search (Px, ExtensionsOfPx, minutil);
15     end
16 end
```

number of transactions containing $X \cup \{z\}$ can only be smaller than the number of transactions containing X. Because of this and that z is a negative item, the utility of $up(X \cup \{z\})$ can only be the same or less than that of $up(X)$. But note that $u(X)$ may be smaller, greater or equal to $u(X \cup \{z\})$.

The previous property can be generalized for successive extensions of an itemset with negative items

Property 10 (downward closure of transitive extensions with negative items). Let X be an itemset. For any itemset Y resulting from transitive extensions of X with negative items, $u(up(Y)) \leq u(up(X))$.

Based on this observation, we can use the following pruning condition.

Property 11 (pruning condition for itemsets containing negative items based on the total order $\succ$). Let X be an itemset such that $up(X) < minutil$ and only negative items can be used to extend X based on the total order $\succ$. Therefore, all transitive extensions of X with these items will be low utility and can be pruned. **Rationale.** This pruning condition directly follows from the previous properties.

But using this pruning condition in the algorithm requires to be able to calculate $u(up(X))$ efficiently. The solution is to separate *iutil* values in utility-list into two values: *iputil* and *inutil*. For a given transaction T_c, *iputil* and *inutil* now respectively indicates $u(up(X), T_c)$ and $u(un(X), T_c)$. Having these

Algorithm 3: The Construct procedure

input : P: an itemset, Px: the extension of P with an item x, Py: the extension of P with an item y
output: the utility-list of Pxy

```
1  UtilityListOfPxy ← ∅;
2  foreach tuple ex ∈ Px.utilitylist do
3      if ∃ey ∈ Py.utilitylist and ex.tid = exy.tid then
4          if P.utilitylist ≠ ∅ then
5              Search element e ∈ P.utilitylist such that e.tid = ex.tid.;
6              exy ← (ex.tid, ex.iutil + ey.iutil − e.iutil, ey.rutil);
7          end
8          else
9              exy ← (ex.tid, ex.iutil + ey.iutil, ey.rutil);
10         end
11         UtilityListOfPxy ← UtilityListOfPxy ∪ {exy};
12     end
13 end
14 return UtilityListPxy;
```

values, $u(up(X))$ can be easily computed and also $u(X)$ by respectively summing the *iputil* values, and summing both the *iputil* and *inutil* values.

Based on the above ideas, the FHN algorithm is obtained by making the following modifications. First, instead of calculating the original TWU, the redefined TWU is used to avoid underestimating the utility of HUIs containing positive items (similarly to HUINIV-Mine presented in section 2). Fig. 2 shows the redefined transaction utility values (left), TWU of single items (center) and EUCS (right), when this modification is done. Second, utility-lists are redefined such that *iputil* and *inutil* elements are used. Furthermore, only utility values of positive items are included in *rutil* values of utility-lists. The reason is that the algorithm can miss some HUIs if *rutil* values of negative items are included in utility lists as we have demonstrated in the last paragraph of Section 2. Third, the $\succ$ total order is defined such that all negative items succeed positive items (as previously explained). Fourth, the TWU pruning condition $TWU(\{x, y\}) < minutil$ using the EUCS structure is only used for positive items. Fifth, the pruning condition described in 11 is the only pruning condition used for deciding whether an itemset should be extended with negative items. Sixth, the pruning condition for positive items based on the sum of *iutil* and *rutil* values is redefined as the sum of *iputil* and *rutil* values.

We now discuss the correctness of these modifications for finding all HUIs when positive and negative items are used. This explanation can be broken down into two parts (1) the algorithm first extends an itemset by appending positive items and (2) then the algorithm appends negative items (based on $\succ$). During the first part, FHN is correct since it behaves as a regular HUI mining algorithm for discovering HUIs containing only positive items. This is true because negative

items are always appended after positive items (thus negative items are not considered when forming HUIs containing only positive items). Furthermore, the pruning condition that the sum of *rutil* and *iputil* values must be higher than minutil is correct since *rutil* values of negative items are not considered in utility-lists when the algorithm is generating HUIs containing only positive items. The pruning condition that an extension P_{xy} should not be explored if $TWU(x, y) < minutil$ also remains correct since the redefined TWU is used. In the second part (when negative items are appended), only the pruning condition based on Property 10 is used. Thus, no HUIs containing negative items should be missed.

4 Experimental Study

We evaluated the performance of the proposed FHN algorithm. Experiments were performed on a computer with a third generation 64 bit Core i5 processor running Windows 7 and 5 GB of free RAM. We compared the performance of FHN with the state-of-the-art algorithm HUINIV-Mine for high-utility itemset mining with negative unit profit. All memory measurements were done using the Java API. Experiments were carried on six real-life datasets having varied characteristics.

- *mushroom* is a dense dataset with 120 distinct items, 8,124 transactions, and an average transaction length of 23 items.
- The *retail* dataset contains 88,162 transactions with 16,470 distinct items and an average transaction length of 10,30 items.
- *kosarak* is a dataset that contains 41,270 distinct items, 990,000 transactions, and transaction have an average length of 8.09 items.
- The *chess* dataset contains 3,196 transactions with 75 distinct items and an average transaction length of 35 items.
- The *psumb* dataset contains 49,046 transactions with 7,116 distinct items and an average transaction length of 74 items.
- The *accidents* dataset contains 340,183 transactions having an average length of 33.80 items, and 468 distinct items.

For all datasets, external utilities for items are generated between -1,000 and 1,000 by using a log-normal distribution and quantities of items are generated randomly between 1 and 5, similarly to the settings of [3, 4, 9, 12]. The source code of all algorithms and datasets can be downloaded from the SPMF data mining library (`http://www.philippe-fournier-viger.com/spmf/`).

For each dataset, we ran the FHN and HUINIV-Mine algorithms, while decreasing the *minutil* threshold until the algorithms became too long to execute, ran out of memory or a clear winner was observed. For each dataset, we recorded the execution time and the maximum memory usage. The comparison of execution times is shown in Fig. 3 for all datasets. For *mushroom*, *retail*, *kosarak*, *chess*, *psumb* and *accidents*, FHN was respectively up to 42 times faster, 18 times faster, 38 times faster, 500 times, 15 times and 25 times faster than HUINIV-Mine.

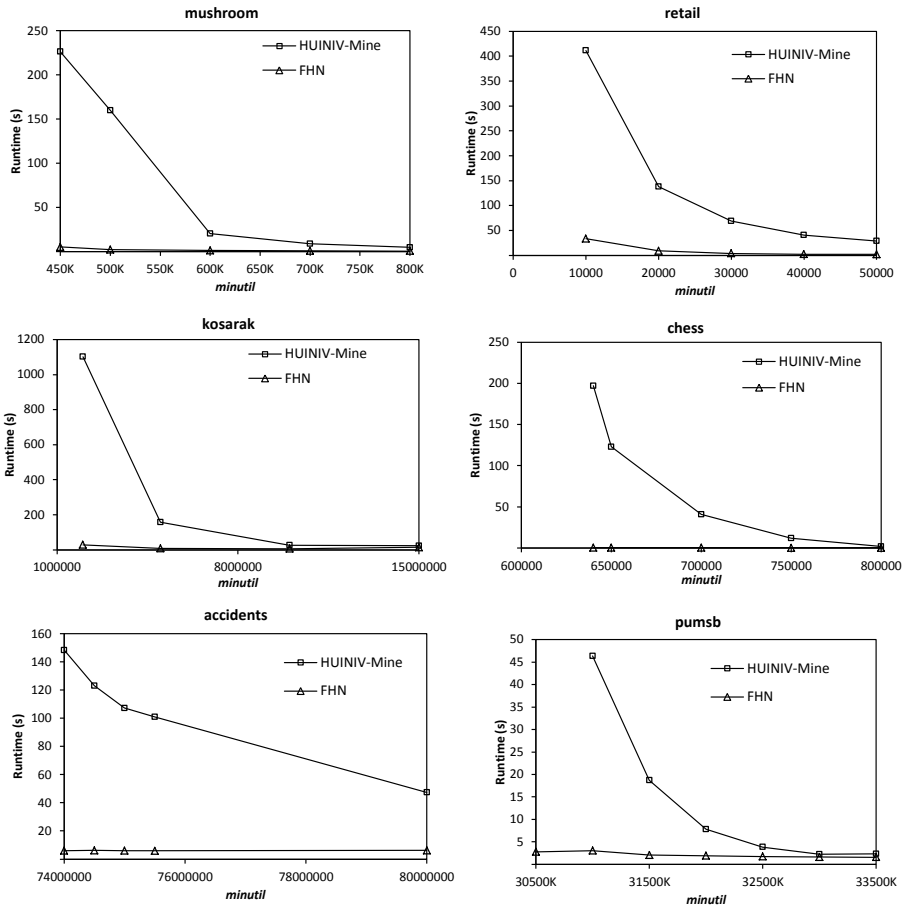

Fig. 3. Execution times

In terms of memory usage, FHN uses much less memory than HUINIV-Mine. For the *mushroom* dataset and *minutil* = 450000, HUINIV-Mine used up to 4.97 GB while FHN was using only up to 250 MB. On *kosarak*, *chess*, *psumb* and *accidents* HUINIV-Mine ran out of memory under our 5 GB memory limit while FHN was respectively using 20 MB, 1179 MB, 100 MB and 350 MB for the lowest *minutil* values. Lastly, for the *retail* dataset, the memory usage of FHN was about five times less than HUINIV-Mine. Overall, FHN used up to 250 times less memory than HUINIV-Mine.

An interesting observation is that FHN performs very well on dense datasets such as *mushroom* compared to HUINIV-Mine. There are several reasons why FHN performs better than HUINIV-Mine. The first reason is that HUINIV-Mine strictly relies on the TWU model for pruning the search space. But the TWU model provides a less strict upper bound on the utility of itemsets than utility-

lists [4, 9]. FHN uses both utility-lists and TWU of itemsets containing two items (the EUCS) to prune the search space. Thus, it can prune a larger part of the search space. The second reason is that defining the total order $\succ$ such that negative items are used to extend itemsets after all positive items have been considered allows handling negative items much more efficiently (when negative items are used to extend an itemset, the exact utility is used to prune the search space). This greatly reduces the search space. Third, HUINIV-Mine is a level-wise algorithm that need to maintain a large amount of itemsets in memory to find larger patterns. Furthermore, since it is using a two-phase approach based on the TWU model it suffers from the problem of generating and maintaining a huge amount of candidates in memory before low-utilty itemsets can be pruned. In FHN, these problems are avoided by using a depth-first search that mines high-utility itemsets without generating candidates. This is what allows FHN to consume much less memory than HUINIV-Mine.

5 Conclusion

In this paper, we have presented a novel algorithm named FHN (Fast High-utility itemset miner with Negative unit profits) for mining HUIs in databases where item unit profits may be positive or negative. The algorithm is an extension of the FHM algorithm [4] for HUI mining.

It is important to note that the strategies that we have presented in the FHN algorithm to handle items with negative unit profits could also be applied in other algorithms that are based on the utility-list structure (e.g. in GHUI-Miner and HUG-Miner [7]).

We have performed an extensive experimental study on six real-life datasets to compare the performance of FHN with the state-of-the-art algorithm HUINIV-Mine. Results show that FHN is up to 500 times faster and can use up to 250 times less memory than HUINIV-Mine, and was shown to perform very well on dense datasets. The source code of all algorithms and datasets used in our experiments can be downloaded as part of the SPMF data mining library `http://www.philippe-fournier-viger/spmf/`. For future work, we are interested in exploring other interesting problems involving utility mining in itemset mining and sequential pattern mining [5, 6].

Acknowledgement This work is financed by a National Science and Engineering Research Council (NSERC) of Canada research grant. Thanks also to Chun-Wei Lin and Wensheng Gan for feedback on the paper.

References

1. Chu, C.-J., Tseng, V. S., Liang, T.: An efficient algorithm for mining high utility itemsets with negative item values in large databases. In: Applied Math. Comput., 215, pp. 767-778 (2009)

2. Agrawal, R., Srikant, R.: Fast algorithms for mining association rules in large databases. In: Proc. Int. Conf. Very Large Databases, pp. 487–499 (1994)
3. Ahmed, C. F., Tanbeer, S. K., Jeong, B.-S., Lee, Y.-K.: Efficient Tree Structures for High-utility Pattern Mining in Incremental Databases. In: IEEE Trans. Knowl. Data Eng. 21(12), pp. 1708–1721 (2009)
4. Fournier-Viger, P., Wu, C.-W., Zida, S., Tseng, V. S.: FHM: Faster High-Utility Itemset Mining using Estimated Utility Co-occurrence Pruning. In: Proc. 21st Intern. Symp. Methodologies Intell. Systems (ISMIS 2014), Springer, pp. 83-92 (2014)
5. Fournier-Viger, P., Gomariz, A., Campos, M., Thomas, R.: Fast Vertical Sequential Pattern Mining Using Co-occurrence Information. In: Proc. 18th Pacific-Asia Conference on Knowledge Discovery and Data Mining, Springer, LNAI, (2014)
6. Fournier-Viger, P., Wu, C.-W., Gomariz, A., Tseng, V. S.: VMSP: Efficient Vertical Mining of Maximal Sequential Patterns. In: Proc. 27th Canadian Conference on Artificial Intelligence, Springer, LNAI, pp. 83-94 (2014)
7. Fournier-Viger, P., Wu, C.-W., Tseng, V. S.: Novel Concise Representations of High Utility Itemsets using Generator Patterns. In: Proc. 10th International Conference on Advanced Data Mining and Applications, Springer LNAI, 14 pages (2014).
8. Li, Y.-C., Yeh, J.-S., Chang, C.-C.: Isolated items discarding strategy for discovering high utility itemsets. In: Data & Knowledge Engineering. 64(1), pp. 198–217 (2008)
9. Liu, M., Qu, J.:Mining High Utility Itemsets without Candidate Generation. In Proceedings of CIKM12, pp. 55–64 (2012)
10. Liu, Y., Liao, W., Choudhary, A.: A two-phase algorithm for fast discovery of high utility itemsets. In: Proc. PAKDD 2005, pp. 689–695 (2005)
11. Shie, B.-E., Cheng, J.-H., Chuang, K.-T., Tseng, V. S.: A One-Phase Method for Mining High Utility Mobile Sequential Patterns in Mobile Commerce Environments. In: Proceedings of IEA/AIE12, pp. 616–626 (2012)
12. Tseng, V. S., Shie, B.-E., Wu, C.-W., Yu., P. S.: Efficient Algorithms for Mining High Utility Itemsets from Transactional Databases. In: IEEE Trans. Knowl. Data Eng. 25(8), pp. 1772–1786 (2013)
13. Wu, C.-W., Fournier-Viger, P., Yu., P. S, Tseng, V. S.: Efficient Mining of a Concise and Lossless Representation of High Utility Itemsets. In: Proceedings of ICDM11, pp. 824–833 (2011)
14. Wu, C.-W., Lin, Y.-F., Yu, P. S., Tseng, V. S.: Mining High Utility Episodes in Complex Event Sequences. In: Proceedings of ACM SIG KDD13, pp. 536–544 (2013)
15. Yin, J., Zheng, Z., Cao, L.: USpan: An Efficient Algorithm for Mining High Utility Sequential Patterns. In: Proceedings of ACM SIG KDD12, pp. 660–668 (2012)
16. Yin, J., Zheng, Z., Cao, L., Song, Y., Wei, W.: Efficiently Mining Top-K High Utility Sequential Patterns. In: Proceedings of ICDM13, pp. 1259–1264 (2013)