FOSHU: Faster On-Shelf High Utility Itemset Mining – with or without Negative Unit Profit

Philippe Fournier-Viger
University of Moncton
18 Antonine-Maillet Ave
Moncton, NB, Canada
philippe.fournier-viger@umoncton.ca

Souleymane Zida
University of Moncton
18 Antonine-Maillet Ave
Moncton, NB, Canada
esz2233@umoncton.ca

ABSTRACT

High utility itemset (HUI) mining is a popular data mining task, which consists of discovering sets of items generating high profit in a transaction database. Recently, several efficient algorithms have been proposed for this task. But, most of them do not consider the on-shelf time periods of items, which thus lead to a bias toward items having more shelf time. Moreover, most algorithms cannot handle databases containing items with a negative unit profit, although this case is very common in real transaction databases. In this paper, we address both of these challenges by proposing a novel efficient algorithm named FOSHU (Faster On-Shelf High Utility itemset miner) to mine HUIs while considering on-shelf time periods of items, and items having positive and/or negative unit profit. An extensive experimental study with real-life datasets shows that the proposed algorithm can be up more than 1000 times faster and use up to 10 times less memory than the state-of-the-art algorithm TS-HOUN for this task. Moreover, experiments show that the proposed algorithm performs well on dense database and databases containing many time periods.

Categories and Subject Descriptors

H.2.8 [**Database Applications**]: Data Mining

Keywords

frequent pattern mining, utility mining, high-utility itemset mining, on-shelf time periods

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, 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). To address these limitations, the problem of FIM has been redefined as *High-Utility Itemset Mining* (HUIM) to consider the more general case where items can appear more than once in each transaction and where each item has a weight (e.g. unit profit) [3, 4, 6, 11, 12, 13, 14, 15]. The goal of HUIM is to discover high utility itemsets (HUIs), i.e. itemsets generating a high profit rather than frequent itemsets. 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, 4, 6, 12, 13, 15]. However, most of these studies make two assumptions that are unrealistic for real transactional databases. First, most algorithms cannot handle databases where items may have negative unit profit/weight. But such items often occur in real-life transaction databases. For example, it is common that a retail store will sell items at a loss to stimulate the sale of other related items or simply to attract customers to their retail location. It was demonstrated that if classical HUIM algorithms are applied on database containing items with negative unit profit, they can generate an incomplete set of HUIs [1, 7]. Second, most algorithms consider that items have the same shelf time, i.e. that all item are on sale for the same time period. However, in real-life some items are only sold during some short time period (e.g. the summer). Algorithms ignoring the shelf time of items have a bias toward items having more shelf time since they have more chance to generate a high profit [9, 10].

To address these respective limitations the problem of HUIM has been respectively redefined as the problem of mining HUIs with negative/positive unit profit [1, 7], and the problem of mining high on-shelf utility itemsets (HOUs) [9]. Recently, a more general problem definition addressing both limitations has been proposed. It is the problem of mining HOUs with negative/positive unit profit [10]. The state-of-the-art algorithm for this latter problem definition is TS-HOUN [9, 10]. Although this algorithm is pioneer, it does not perform well on large or dense databases (as shown in our experiment). The reason is that it uses a three phase approach that requires generating and main-

SAC'15 April 13-17, 2015, Salamanca, Spain.
 ACM 978-1-4503-3196-8/15/04 15.00
http://dx.doi.org/10.1145/2695664.2695823

Table 1: A Transaction Database

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

taining large amount of candidates in memory and perform multiple database scans. Furthermore, its performance deteriorates when the number of time periods in a database is large since TS-HOUN mines patterns in each time period separately and then perform a costly merge operations of results found in each time period.

In this paper, we address these issues by presenting a novel algorithm named FOSHU (Faster On-Shelf High Utility itemset miner) to mine HUIs while considering both positive and negative item unit profit and the items' on-shelf time periods. FOSHU discovers itemsets in a single phase without generating candidates, and mines all times periods at the same time, thus avoiding the costly merge operations of patterns found in each time period. Furthermore, FOSHU introduces novel strategies to handle negative values efficiently. We compare the performance of FOSHU and TS-HOUN on several real-life datasets. Results show that FOSHU can be more than 1000 times faster and uses up to 10 times less memory than TS-HOUN. Furthermore, experiments show that FOSHU performs very well on dense databases and databases with many time periods.

The rest of this paper is organized as follows. Section 2, 3, 4 and 5 respectively presents the problem definition and related work, the FOSHU algorithm, the experimental evaluation and the conclusion.

2. PROBLEM DEFINITION AND RELATED WORK

We first first introduce the problem definition.

Definition 1. Let I be a set of items. Let D be a *transaction database* containing 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*, or *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, or its *purchase quantity*. Let PE be a set of positive integers representing time periods. Each transaction $T_C \in D$ is associated to a time period $pt(T_c) \in PE$, representing the time period during which the transaction occurred.

Example 1. Consider the transaction database shown in Table 1, which will be the running example. This database contains five transactions $(T_1, T_2...T_5)$ and three time periods $(1, 2, 3)$. Transaction T_2 occurred in time period 1, and contains items a, c, e and g, which respectively appear in T_2 with an internal utility of 2, 6, 2 and 5. Table 2 indicates that the external utility of these items are respectively -5, 1, 3 and 1. Thus, item a is sold at loss.

Table 2: External Utility Values (Unit Profit)

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

Definition 2. The *utility of an item $i \in I$ 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 defined as $u(X, T_c) = \sum_{i \in X} u(i, T_c)$. It represents the profit generated by items in X in transaction T_c.

Definition 3. The *time periods (shelf time) of an itemset $X \subseteq I$*, is the set of time periods where X was sold, defined as $pi(X) = \{pt(T_c) | T_c \in D \wedge X \subseteq T_c\}$.

Definition 4. The *utility of an itemset $X \subseteq I$ in a time period $h \in pi(X)$* is denoted as $u(X, h)$ and defined as $u(X, h) = \sum_{T_c \in D \wedge h = pt(T_c)} u(X, T_c)$. The *utility of an itemset $X \subseteq I$ in a database D* is defined as $u(X) = \sum_{h \in pi(X)} u(X, h)$.

Example 2. The utility of item e in T_3 is $u(e, T_2) = 3 \times 2 = 6$. The utility of the itemset $\{c, e\}$ in T_2 is $u(\{c, e\}, T_2) = u(c, T_2) + u(e, T_2) = 1 \times 6 + 3 \times 2 = 12$. The time periods of itemset $\{c, e\}$ are $pi(\{c, e\}) = \{1, 2, 3\}$. The utility of $\{c, e\}$ in periods 1, 2 and 3 are respectively $u(\{c, e\}, 1) = 12$, $u(\{c, e\}, 2) = 4$ and $u(\{c, e\}, 3) = 11$. The utility of $\{c, e\}$ in the database is $u(\{c, e\}) = 12 + 4 + 11 = 27$.

Definition 5. 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. $TU(T_c) = \sum_{i \in T_c} u(i, T_c)$. Given an itemset X, the *total utility of the time periods of X* is defined as $to(X) = \sum_{h \in pi(X) \wedge T_c \in D \wedge h = pt(T_c)} TU(T_c)$. The *relative utility of an itemset $X \subseteq I$ in a database D* is defined as $ru(X) = u(X)/|to(X)|$ if $to(X) \neq 0$, and is defined as 0 otherwise. The relative utility of an itemset X represents how large the profit/loss generated by X is compared to the total profit/loss generated during the time periods where X was sold. The relative utility measure is useful for retailers as it is an indicator of the relative selling performance (profit/loss) of an itemset during time periods where it was on the shelves. It can thus be used to compare the selling performance of various itemsets in terms of their relative contribution to the overall profit of a retail store, to determine which itemsets are relatively the most profitable when they are sold.

Example 3. The transaction utility of transactions T_1, T_2, ... T_5 are respectively $TU(T_1) = -2$, $TU(T_2) = 7$, $TU(T_3) = 20$, $TU(T_4) = 20$ and $TU(T_5) = 11$. The total utility of the time periods of $\{c, e\}$ is $to(\{c, e\}) = 58$. The relative utility of $\{c, e\}$ is $ru(\{c, e\} = u(\{c, e\})/to(\{c, e\}) = 27/58 = 0.46$.

Definition 6. An itemset X is a *high on-shelf utility itemset* (HOU) if its relative utility $ru(X)$ is no less than a user-specified minimum utility threshold *minutil* given by the user ($0 \geq minutil \geq 1$). Otherwise, X is a *low on-shelf utility itemset*. The *problem of high on-shelf utility itemset mining* is to discover all HOUs in a database where external utility values are positive [9]. The *problem of high on-shelf utility itemset mining with negative/positive unit profit* [10] is to discover all HOUs in a database where external utility values may be positive or negative.

Example 4. If $minutil = 0.43$, 22 HOUs are found in the database of the running example. They are $\{a, b, c, d, e, f\}$:0.44, $\{b, d, f\}$:0.47, $\{b, d, e, f\}$:0.53, $\{b, c, d, e, f\}$:0.55, $\{b, c, d, e\}$:0.49, $\{d, e, f\}$:0.44, $\{c, d, e, f\}$:0.47, $\{b, g\}$:0.54, $\{b, e, f\}$:0.81, $\{b,c, e, g\}$:1.0, $\{b, c, g\}$:0.72, $\{e, g\}$:0.51, $\{c, e, g\}$:0.77, $\{c, g\}$:0.48, $\{b, d\}$:0.67, $\{b, d, e\}$:0.8, $\{b, c, d, e\}$:0.89, $\{b, c, d\}$:0.75, $\{c, d, e\}$:0.43, $\{b, e\}$:0.45, $\{b, c, e\}$:0.55 and $\{d, e\}$:0.53, where the relative utility of each itemset is indicated after a colon.

It can be demonstrated that the (relative) utility measure is not monotonic or anti-monotonic [9, 10, 12, 13, 15]. 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 HOUs. The state-of-the-art algorithm for HOU mining is *TS-HOUN* [9, 10]. It circumvents the previous problem by overestimating the utility of itemsets using a measure called the Transaction-Weighted Utilization (TWU) [3, 13, 15], which is anti-monotonic. The TWU measure assumes that all items have positive external utility values. The TWU measure is defined as follows.

Definition 7. For a given time period h, the *transaction-weighted utilization* (TWU) of an itemset X is denoted as $TWU(X, h)$ and defined as the sum of the transaction utility of transactions from h containing X, i.e. $TWU(X, h) = \sum_{T_c \in D \wedge X \subseteq T_c \wedge pt(T_c)=h} TU(T_c)$.

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 3, 17, 25, 20 and 11. $TWU(\{b\}, 2) = TU(T_3)+TU(T_4) = 25+20 = 45$. $TWU(\{b\}, 3) = TU(T_5) = 11$.

The TWU measure has three important properties that are used to prune the search space in TS-HOUN [10, 13]. These properties only hold if external utility values of items are positive [1].

Definition 8. The utility of a time period h is defined as $pto(h) = \sum_{T_c \in D \wedge h=pt(T_c)} TU(T_c)$. The relative utility of an itemset X for a time period h is $ru(X, h) = u(X, h)/pto(h)$.

Property 1. The TWU of an itemset X for a period h is an upper bound on the utility of X in period h, i.e. $TWU(X, h) \geq u(X, h)$.

Property 2. For a period h, the TWU measure is anti-monotonic. Let X and Y be two itemsets. If $X \subset Y$, then $TWU(X, h) \geq TWU(Y, h)$

Property 3. The TWU of an itemset X for a period h divided by the total utility of the time period h is an upper bound on the relative utility of X in period h, i.e. $TWU(X, h)/pto(h) \geq ru(X, h)$.

Property 4. Let X be an itemset. If there exists no time period h such that $TWU(X, h)/pto(h) \geq minutil$, then X is not a high on-shelf utility itemset. Otherwise, X may or may not be a high on-shelf utility itemset.

TS-HOUN is a three phrase algorithm, which works similarly to other multiple-phase algorithms for HUI mining (e.g. Two-Phase [13], IHUP [3] and UPGrowth [15]). In *Phase 1*, TS-HOUN scans the database to calculate the transaction utility of each transaction, and group transactions by time period. Then, for each time period h, it calculates the TWU of all single item and pairs of items. For each time period, TS-HOUN keep each itemset X such that $TWU(X)/pto(h) \geq minutil$. The union of itemsets found in all time periods is then calculated since only those itemsets may be part of a HOU (by Property 4). In *Phase 2*, those itemsets are used to generate larger itemsets in each time period by applying a modified Two-Phase [13] HUI mining algorithm, which explores itemsets in a level-wise manner similar to Apriori [2] but using pruning Property 2. Then, the union of the *candidates* found in each time period is calculated. Finally, in *Phase 3*, TS-HOUN scans the database again to calculate the relative utility of each candidate and keep only HOUs. The TS-HOUN algorithm as described above is complete only if applied on a database where items have positive unit profit [10]. To address the problem of mining HOUs with items having negative unit profits, the authors of TS-HOUN redefined the TWU as follows based on previous work from Chu et al [1].

Definition 9. The *redefined transaction utility* (RTU) of a transaction T_c is the sum of the utility of the items from T_c having positive external utilities, that is $RTU(T_c) = \sum_{x \in T_c \wedge p(x)>0} u(x, T_c)$. The *redefined transaction-weighted utilization* (RTWU) of an itemset X for a time period h is defined as $RTWU(X, h) = \sum_{T_c \in D \wedge X \subseteq T_c \wedge pt(T_c)=h} RTU(T_c)$.

Using RTWU instead of TWU has the effect of restoring Property 4 and this is what allows TS-HOUN to find the complete set of HOUs.

Although, TS-HOUN is a pioneer algorithm for mining HOUs in a database with negative unit profits and time periods, it is inefficient for several reasons. First, it generate candidates by mining each time period separately. Therefore, it may have to generate the same itemset several times in different periods before performing a costly union operation of itemsets in all time periods. Another problem is that TS-HOUN has to maintain a huge amount of candidates in memory. Furthermore, TS-HOUN suffers from the well-known problems of level-wise approach for itemset mining such as generating candidates not appearing in the database [2].

To address this issue, in this paper, we propose a novel algorithm that can mine HOUs using a single phase, without maintaining candidates in memory, and that mine all time periods at the same time rather than mining each time period separately. The proposed algorithm is inspired by FHM [4], a recently proposed algorithm for high utility itemset mining, which is to our knowledge the fastest HUI mining algorithm. FHM is designed to handle only positive external utility values and does not consider time periods. 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 [13], UPGrowth [15] and IHUP [3]. FHM utilizes a depth-first search to explore the search space of itemsets. FHM associates a structure to each pattern named *utility-list*, previously introduced in the HUI-Miner [12] algorithm. 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 10. 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 6. 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 $\{(T_1, 11, 0), (T_3, 17, 8)\}$.

To discover HUIs, FHM performs 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 calculating an upper-bound on the utility of its supersets to prune the search space is done based on the two following properties

Property 5. Let be an itemset X. The utility $u(X)$ is equal to the sum of *iutil* values in $ul(X)$ [12].

Property 6. Let be an itemset X. The sum of *iutil* and *rutil* values in $ul(X)$ is an upper bound on $u(X)$. Moreover, it can be shown that this upper bound is tighter than TWU(X). [12].

Property 7. 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 $y \succ i$, $\forall i \in X$. The utility of transitive extensions of X can only be lower or equal to the the sum of *iutil* and *rutil* values in $ul(X)$. [12].

Before presenting our algorithm, we demonstrate with an example that the property above used by FHM to prune the search space is invalid if negative external utility values appears in a database. Consider that items a and d in the running example have respectively external utility values of 5 and -2 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, none of its supersets can have a utility higher than 6. However, this is not the case since itemset $u(\{a, b, f\}) = 14$.

3. THE FOSHU ALGORITHM

In this section, we present our proposal, the FOSHU algorithm. It relies on the utility-list structure used in the FHM [4] algorithm and the TWU measure, but it also introduces several novel ideas to handle time-periods and items with negative unit profit.

In the next subsections, we first address the problem of generalizing properties of utility-lists for time periods. Then, we present the main procedure of the algorithm. The result is an algorithm for HOU mining. Finally, we explain how this algorithm is extended to also handle items with negative unit profits.

3.1 Handling time periods

We adapt properties of utility lists to databases containing time periods as follows.

Property 8. Let X be an itemset and h be a time period. Let $sumIUtil(X, h)$ and $sumRUtil(X, h)$ respectively denote the sum of *iutil* and *rutil* values in the utility list $ul(X)$ for the time period h. An upper bound on $u(X)$ is $sumIUtil(X, h)+sumRUtil(X, h)$, that is $sumIUtil(X, h)+sumRUtil(X, h) \geq u(X, h)$.

Property 9. Let X be an itemset and h be a time period. The inequation $sumIUtil(X, h) + sumRUtil(X, h) \leq TWU(X, h)$ holds, that is $sumIUtil(X, h)+sumRUtil(X, h)$ is a tighter upper bound on $u(X, h)$ than $TWU(X, h)$.

Property 10. Let X and Y be two itemsets. If $X \subset Y$, then $sumIUtil(X, h)+sumRUtil(X, h) \geq sumIUtil(Y, h)+sumRUtil(Y, h)$.

Property 11. For an itemset X and a period h, the value $sumIUtil(X, h)+sumRUtil(X, h)$ divided by the total utility of the time period $pto(h)$ is an upper bound on the relative utility of X in period h, i.e. $(sumIUtil(X, h) + sumRUtil(X, h))/pto(h) \geq ru(X, h)$.

Property 12. Let X be an itemset. If there exists no time period h such that $sumIUtil(X, h)+sumRUtil(X, h)/pto(h) \geq minutil$, then X is not a high on-shelf utility itemset as well as any transitive extensions of X according to the total order $\succ$. Recall that an extension of X is an itemset obtained by appending an item y to X such that $y \succ i$, $\forall i \in X$.

Proof of the above properties are not given due to space limitation. Besides, note that for implementation, utility-lists are redefined so that elements in utility lists are grouped by time period. This bring two benefits for efficiency. It allows to calculate $sumIUtil(X, h)$ and $sumRUtil(X, h)$ for a time period h and an itemset X by only scanning elements representing transactions in h in $ul(X)$. Second, it allows a faster construction of utility-lists by only comparing elements of two utility lists if they belong to the same time period.

3.2 Main procedure

The main procedure (Algorithm 1) takes as input a transaction database and the *minutil* threshold. The algorithm first scans the database to calculate the global TWU of each item i, which is defined as $TWU(i) = \sum_{h \in pi(x)} TWU(X, h)$, and $TWU(\{i\}, h)$ for each period h. Moreover, the set of all time periods PE and the utility $pto(h)$ of each period $h \in PE$ is computed during the first database scan.

Then, the algorithm attempts to remove single items that may not be part of a high on-shelf utility itemset. This is

done by computing $to(\{i\})$ for each item i by using $pi(i)$ and the utility of each period previously obtained. This allows to create the set I^* containing all items i such that there exists at least a period h such that $TWU(\{i\}, h)/to(\{i\}) \geq minutil$. Thereafter, all items not in I^* will be ignored since they cannot be part of a high on-shelf utility itemset by Property 3. It is important to note that the TWU needs to be used here to prune single items instead of the sum of *iutil* and *rutil* value of utility-lists (Property 3), since this latter pruning strategy is defined w.r.t extensions of an itemset, but at this stage extensions are not considered yet.

The TWU values of items are then used to establish a total order $\succ$ on the set of items I^*, which is the order of ascending global TWU values. This order is used since it was shown to reduce the search space when using a depth-first exploration in HUI mining [4, 12, 15].

A second database scan is then performed. During this database scan, items in transactions are reordered according to the total order $\succ$ and the utility-list of each item $i \in I^*$ is built. After the construction of utility-lists, the depth-first search exploration of itemsets starts by calling the recursive procedure *Search* with the empty itemset $\emptyset$, the set of items I^* and *minutil*.

Algorithm 1: The FOSHU algorithm

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

1 Scan D to calculate $TWU(\{i\})$, $TWU(\{i\}, h)$ and $pto(h)$ for each period h and each item i, as well as the set of all time periods PE;
2 Let $I^* = \{i | \exists h \in PE \wedge TWU(\{i\}, h) \ /pto(h) \geq minutil\}$;
3 Let $\succ$ be the global TWU ascending order on I^*;
4 Scan D to build the utility-list of each item $i \in I^*$;
5 `Search` ($\emptyset$, I^*, *minutil*);

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, and (3) *minutil*. The search procedure operates as follows. For each extension Px of P, the search procedure first scans the utility list of Px to calculate $sumIUtil(Px, h)$ for each period h where Px appears. At the same time, the total utility of time periods where Px appears ($to(Px)$) is calculated, as well as the total utility of Px (which is equal to $sumIUtil(Px)$ by Property 5). Then, the relative utility of Px is calculated as $ru(Px) = sumIUtil(Px) /to(Px)$. If $ru(Px)$ is no less than *minutil*, Px is a high on-shelf utility itemset and it is output. Then, if there exist a time period h such that the sum of $sumIUtil(Px, h)+ sumRUtil(Px, h))/to(Px)$ is no less than *minutil*, it means that extensions of Px should be explored (by Property 12). This is performed by merging Px with all extensions Py of P such that $y \succ x$ 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 described in more details. Then, a check is performed to determine if Pxy and its extensions may be high on-shelf utility itemsets by using the TWU measure (line 12), based on Property 4. This is done by scanning the utility list of Pxy to calculate $TWU(Pxy, h)$ for each time period h where Pxy appears. If $TWU(Pxy, h)/pto(h) \geq minutil$ for at least one period h, then Pxy will be added to $ExtensionsOfPx$, the set of extensions of Px which will be considered for further extensions with a recursive call to the *Search* procedure. Note that the check at line 12 may seem redundant since the TWU is a less tight upper bound than the sum of *iutil* and *rutil* values used in the check at line 6 (that would be performed in a recursive call to *Search* with Pxy). However, there is a good reason for performing the check at line 12. It is that pruning an itemset Pxy before the recursive call to *Search* will avoid comparing Pxy with potentially many other extensions of Px in the recursive call.

Since the *Search* procedure starts from single items, it recursively explore the search space of itemsets by appending single items, it can be easily seen based on Properties 3, 4, 5 and 12 that this procedure is correct and complete to discover all high utility on-shelf itemsets.

Algorithm 2: The *Search* procedure

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

1 **foreach** *itemset* $Px \in ExtensionsOfP$ **do**
2 Scan $Px.utilitylist$ to calculate $sumIUtil(Px, h)$, $sumRUtil(Px, h)$ for each period h where Px appears, $to(X)$ and $sumIUtil(Px)$;
3 **if** $sumIUtil(Px)/to(X) \geq minutil$ **then**
4 output Px;
5 **end**
6 **if** $\exists h \in pi(Px)$ such that $(sumIUtil(Px, h)+ sumRUtil(Px, h))/to(Px) \geq minutil$ **then**
7 $ExtensionsOfPx \leftarrow \emptyset$;
8 **foreach** itemset $Py \in ExtensionsOfP$ such that $y \succ x$ **do**
9 $Pxy \leftarrow Px \cup Py$;
10 $Pxy.utilitylist \leftarrow$ `Construct` (P, Px, Py);
11 Scan $Pxy.utilitylist$ to calculate $TWU(Pxy, h)$ for each period h where Pxy appears;
12 **if** $\exists h \in pi(Pxy)$ such that $TWU(Pxy, h)/pto(h) \geq minutil$ **then**
13 $ExtensionsOfPx \leftarrow ExtensionsOfPx \cup Pxy$;
14 **end**
15 **end**
16 `Search` (Px, $ExtensionsOfPx$, *minutil*, $pi(Px)$);
17 **end**
18 **end**

3.3 Handling negative unit profit

We now describe how the proposed algorithm is extended to not just handle time periods but also items with negative unit profit. Let the term "positive items" and "negative items" denote items respectively having positive and negative unit profit. To be able to transform the algorithm described in the previous subsection into an algorithm that outputs all HOUs when both negative and positive items are used, we make the following modifications.

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;
```

First, we define the total order $\succ$ such that negative items always succeed all positive items. By using this order, positive items are always use to extend an itemset first before using negative items. This let us define new pruning properties. In the following, for an itemset X we use the notation $up(X)$ and $un(X)$ to respectively refer to the set of all positive items in X and the set of all negative items in X.

Property 13. Let X be an itemset. It follows that $u(X,h) \leq u(up(X),h)$.

Proof. This property holds because $X \setminus up(X) = un(X)$ and negative items can only decrease the utility of X. □

Property 14. Let X be an itemset and z be a negative item such that $z \notin X$. It follows that $u(up(X \cup \{z\}),h) \leq u(up(X),h)$.

Proof. Because z is a negative item, $up(X,h) = up(X \cup \{z\},h)$. Moreover, the number of transactions containing $X \cup \{z\}$ in h can only be smaller than the number of transactions containing X. Thus, $u(up(X \cup \{z\}),h) \leq u(up(X),h)$. But note that $u(X,h)$ may be smaller, greater or equal to $u(X \cup \{z\},h)$. □

Property 15. Let X be an itemset. For any itemset Y resulting from transitive extensions of X with negative items, $u(up(Y),h) \leq u(up(X),h)$.

Proof. This directly follows from previous property. □

Based on this observation, we can use the following property to prune transitive extensions of an itemset with negative items.

Property 16. Let X be an itemset. If only negative items may be appended to X according to $\succ$ and there exists no time period h such that $u(up(X),h) \,/\, pto(h) \geq minutil$, then X is not a high on-shelf utility itemset as well as any transitive extension Y of X.

But integrating this pruning condition in the algorithm requires to be able to calculate $u(up(X),h)$ efficiently. To achieve this, we separate *iutil* values in utility-lists into two values: *iputil* and *inutil*. For a given transaction T_c, *iputil* and *inutil* respectively indicates $u(up(X),T_c)$ and $u(un(X),T_c)$. Having these values, $u(up(X),h)$ can be easily computed and also $u(X,h)$ by respectively summing the *iputil* values (denoted as $sumIPUtil(X,h)$) of period h, and summing both the *iputil* and *inutil* values of period h.

Besides, we also redefine *rutil* values so that only utility values of positive items are considered. The reason is that the algorithm can miss some HOUs if *rutil* values of negative items are included in utility lists as we have demonstrated in the last paragraph of Section 2. Based on the above modifications and definitions, we can rewrite pruning Property 12 as follows.

Property 17. Let X be an itemset. If there exists no time period h such that $sumIPUtil(X,h) + sumRUtil(X,h) \,/\, pto(h) \geq minutil$, then X is not a high on-shelf utility itemset as well as any transitive extensions of X.

Proof. If we only consider transitive extensions with positive items, the property becomes equivalent to Property 12. Now, for extensions with negative items succeeding extensions with positive items according to $\succ$, the property is also true since when negatives items are used for extensions this property becomes Property 16. This is true because $sumRUtil(X,h)$ will be equal to 0 and $sumIPUtil(X,h)$ will be equal to $u(up(X),h)$. □

Based on the above ideas, the modified FOSHU 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 HOUs containing positive items, since it was shown in TS-HOUN [10] that using the redefined TWU will not prune HOUs. 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 (as previously explained). Third, the $\succ$ total order is defined such that all negative items succeed positive items (as previously explained). Fourth, the pruning condition based on Property 12 for positive items based on the sum of *iutil* and *rutil* values is redefined as Property 17.

4. EXPERIMENTAL STUDY

We performed several experiments to assess the performance of the proposed 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 FOSHU with the state-of-the-art algorithm TS-HOUN for high on-shelf utility itemset mining with negative unit profits. All memory measurements were done using the Java API. Experiments were carried on five real-life datasets having varied characteristics. The characteristics of datasets are shown in table 4, where the $|D|$, $|I|$ and $avgL$ columns respectively indicate the number of transactions, the number of distinct items and average transaction length.

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, 12, 15]. Moreover, the

dataset	$\|D\|$	$\|I\|$	$avgL$
mushroom	8,124	120	23
retail	88,162	16,470	10.3
accidents	340,183	468	33.8
chess	3,196	75	35
pumsb	49,046	7,116	74

Table 3: Dataset characteristics

number of time periods was set set to 5 as in [10]. The source code of all algorithms and datasets can be downloaded as part of the SPMF open-source data mining library `http://www.philippe-fournier-viger.com/spmf/` [8].

4.1 Influence of *minutil* threshold

We ran the FOSHU and TS-HOUN algorithms on each dataset while decreasing the *minutil* threshold until 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. 1 for the datasets using a logarithmic scale. For *mushroom*, *retail*, *chess*, *pumsb* and *accidents*, FOSHU clearly outperform TS-HOUN. It is respectively up to 1000 times faster, 683 times faster, 2000 times faster, 89 times, and 178 times faster than TS-HOUN.

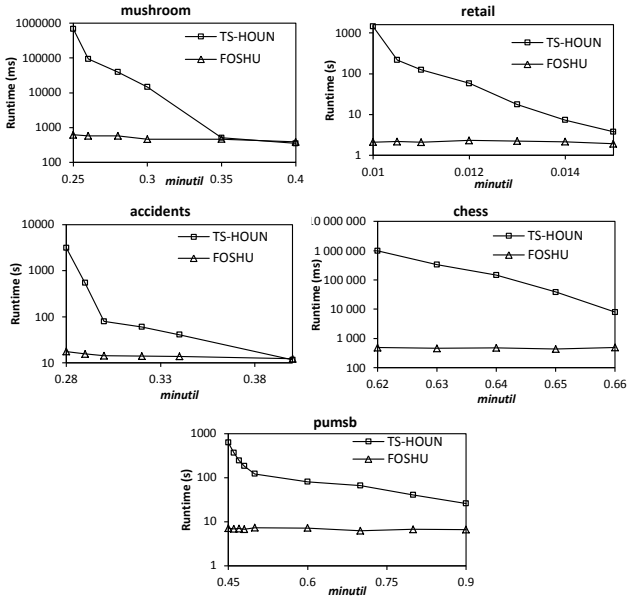

Figure 1: Execution time w.r.t *minutil*

In terms of memory usage, FOSHU has a lower memory consumption on all datasets. For the *mushroom*, *retail*, *accidents*, *chess* and *pumsb*, the memory usage of TS-HOUN and FOSHU for the lowest *minutil* values are respectively 69/39, 571/539, 139/14, 602/498 and 123/98. Overall, FOSHU used up about 10 times less memory than TS-HOUN on some datasets.

There are several reasons why FOSHU has the best overall performance. The first reason is that FOSHU utilizes the sum of *iutil* and *rutil* values as well as the TWU to prune the search space, while TS-HOUN only relies on the TWU. Since the sum of *iutil* and *rutil* values is a stricter upper bound than the TWU, this allows FOSHU to further prune the search space. The second reason is that FOSHU utilizes a very efficient depth-first search that mines high on-shelf itemsets without generating candidates, and mines all time periods at the same time. On the other hand, TS-HOUN is a level-wise algorithm that needs to maintain a large amount of itemsets in memory to find larger patterns, and mine patterns in each time periods separately before merging results of all time periods. Furthermore, since TS-HOUN uses a multiple phase approach, it suffers from the problem of generating and maintaining a huge amount of candidates in memory before low-utilty itemsets can be pruned. The third reason is that negative items are handled more efficiently in FOSHU than in TS-HOUN. FOSHU introduces the use of a total order $\succ$ such that negative items are used to extend itemsets after all positive items have been considered. By replacing the use of *iutil* values by *iputil* and *inutil* values, we obtained a pruning condition based on $sumIPUtil(X, h)$ + $sumRUtil(X, h)$. This condition is more tight than the redefined TWU and is simplified as $sumIPUtil(X, h)$ for extensions with negative items (since negative items are not considered in *rutil* values). Lastly, an interesting observation is that FOSHU performs very well on sparse datasets (e.g. *retail*), as well as dense datasets (e.g. *mushroom* and *chess*).

4.2 Influence of the number of time periods

We also performed an experiment to assess the influence of the number of time periods on the execution time. We ran FOSHU and TS-HOUN on the same datasets but randomly grouped transactions into 5, 25 and 50 time periods. Results are shown in Fig. 2 for the *retail* dataset. As we can see, FOSHU has a much better scalability w.r.t to the number of periods. FOSHU is only 1.26 times slower when the dataset has 50 time periods than when the dataset has 5 time periods, while TS-HOUN is 327 times slower. Results for the other datasets are not shown due to space limitation. But the same trend can be observed. The reason why FOSHU performs better is that it mines all time periods at the same time, while TS-HOUN mines them separately and merge results found in each-time periods, which degrades its performance when the number of time periods is large.

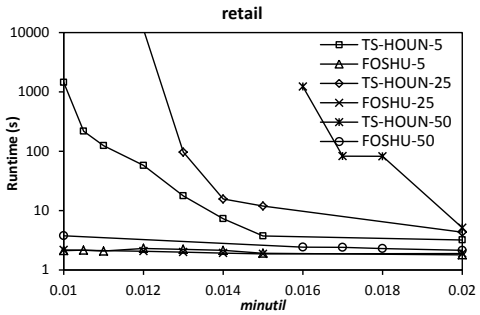

Figure 2: Execution time w.r.t time period count

4.3 Influence of the number of transactions

We also performed an experiment to asses the scalability of FOSHU w.r.t the number of transactions. For this

experiment, we ran FOSHU on all datasets with the lowest *minutil* value used in previous experiments, and varied the number of transactions from 25% to 100%. Results are shown in Table 4.3. We can observe that FOSHU has linear scalability w.r.t to the number of transactions. Scalability of TS-HOUN is also linear and is not shown due to space limitation.

dataset	25%	50%	75%	100%
mushroom	0.6	0.6	0.6	0.6
retail	2.1	2.1	2.1	2.1
accidents	9.3	9.5	9.8	10.2
chess	0.3	0.3	0.3	0.3
pumsb	4.1	4.1	4.2	4.2

Table 4: Scalability of FOSHU w.r.t database size

5. CONCLUSION

In this paper, we have presented a novel algorithm named FOSHU (Fast On-Shelf High Utility itemset miner) for mining high utility itemsets in databases where negative external utility values appear and shelf time of items are considered. FOSHU brings several improvements over previous algorithms. By using utility-lists, it is a single phase algorithm that does not need to maintain candidates in memory. It also relies on a depth-first search rather than a level-wise search. Moreover, FOSHU mines HOUs in all time periods at the same time rather than mining each period separately and performing costly post-processing of the results of each time period. FOSHU also introduces the novel idea of using a total order where negative items are last to handle items with negative external utility values more efficiently.

An extensive experimental study with five real-life datasets shows that FOSHU can be more than three orders of magnitude faster and can use up to 10 times less memory than the state-of-the-art algorithm TS-HOUN, and was shown to perform very well on dense datasets. Furthermore, experiments show that FOSHU performs very well on dense databases and databases with many time periods. The source code of all algorithms and datasets used in our experiments can be downloaded as part of the open-source SPMF data mining library `http://www.philippe-fournier-viger.com/spmf/` [8]. For future work, we are interested in exploring other interesting problems involving utility in itemset mining such as sequential pattern mining [5].

6. ACKNOWLEDGEMENTS

This work is financed by a National Science and Engineering Research Council (NSERC) of Canada research grant.

7. REFERENCES

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

[2] R. Agrawal and R. Srikant. Fast algorithms for mining association rules in large databases. In *Proc. Int. Conf. Very Large Databases*, pp. 487–499, 1994.

[3] C.F. Ahmed, S.K. Tanbeer, B.-S. Jeong and Y.-K. Lee. Efficient Tree Structures for High-utility Pattern Mining in Incremental Databases. In *IEEE Trans. Knowl. Data Eng*, 21(12):1708–1721, 2009.

[4] P. Fournier-Viger, C.-W. Wu, S. Zida and V.S. Tseng. Faster High-Utility Itemset Mining using Estimated Utility Co-occurrence Pruning. In *Proc. 21st Intern. Symp. Methodologies Intell. Systems*, Springer, pp. 83–92, 2014.

[5] P. Fournier-Viger, A. Gomariz, M. Campos and R. Thomas. Fast Vertical Sequential Pattern Mining Using Co-occurrence Information. In *Proc. 18th Pacific-Asia Conference Knowl. Discovery and Data Mining*, Springer, LNAI, pp. 40–52, 2014.

[6] P. Fournier-Viger, C.W. Wu and V.S. Tseng. Novel Concise Representations of High Utility Itemsets using Generator Patterns. In *Proc. 10th International Conference on Advanced Data Mining and Applications*, Springer, 2014.

[7] P. Fournier-Viger. FHN: Efficient Mining of High-Utility Itemsets with Negative Unit Profits. In *Proc. 10th International Conference on Advanced Data Mining and Applications*, Springer, 2014.

[8] P. Fournier-Viger, A. Gomariz, A. Soltani, T. Gueniche, C.W. Wu., V.S. Tseng. SPMF: a Java Open-Source Pattern Mining Library. In *Journal of Machine Learning Research*. 15:,2014.

[9] G.-C. Lan, T.-P. Hong and V.S. Tseng. Discovery of high utility itemsets from on-shelf time periods of products. In *Expert Systems with Applications*. 38:5851–5857, 2011.

[10] G.-C. Lan, T.-P. Hong, J.-P. Huang and V.S. Tseng. On-shelf utility mining with negative item values. In *Expert Systems with Applications*. 41:3450–3459, 2014.

[11] C.-Y. Li, J.-S. Yeh, C.-C. Chang. Isolated items discarding strategy for discovering high utility itemsets. In *Data & Knowledge Engineering*, 64(1): 198–217, 2008.

[12] M. Liu and J. Qu. Mining High Utility Itemsets without Candidate Generation. In *Proc. 21st ACM Intern. Conf. Inform. Known. Management*, pp. 55–64, 2012.

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

[14] B.-E. Shie, J.-H. Cheng, K.-T. Chuang and V.S. Tseng. A One-Phase Method for Mining High Utility Mobile Sequential Patterns in Mobile Commerce Environments. In *Proc. 25th Intern. Conf. Indust., Eng and Other Applications of Applied Intelligent Systems*, pp. 616–626, 2012.

[15] V.S. Tseng, B.-E. Shie, C.-W. Wu and P.S. Yu. Efficient Algorithms for Mining High Utility Itemsets from Transactional Databases. In *IEEE Trans. Knowl. Data Eng*, 25(8):1772–1786, 2013.

[16] J. Yin, Z. Zheng, L. Cao. USpan: An Efficient Algorithm for Mining High Utility Sequential Patterns. In *Proc. 8th ACM SIGKDD Intern. Conf. Knowl. Discovery and Data Mining*, pp. 660–668, 2012.