FHM+: Faster High-Utility Itemset Mining using Length Upper-Bound Reduction

Philippe Fournier-Viger[1], Jerry Chun-Wei Lin[2], Quang-Huy Duong[3], Thu-Lan Dam[3,4],

[1] School of Natural Sciences and Humanities, Harbin Institute of Technology Shenzhen Graduate School, China
[2] School of Computer Science and Technology, Harbin Institute of Technology Shenzhen Graduate School, China
[3] College of Computer Science and Electronic Engineering, Hunan University, China
[4] Faculty of Information Technology, Hanoi University of Industry, Vietnam
philfv@hitsz.edu.cn, jerrylin@ieee.org, huydqyb@gmail.com, lanfict@gmail.com

Abstract. High-utility itemset (HUI) mining is a popular data mining task, consisting of enumerating all groups of items that yield a high profit in a customer transaction database. However, an important issue with traditional HUI mining algorithms is that they tend to find itemsets having many items. But those itemsets are often rare, and thus may be less interesting than smaller itemsets for users. In this paper, we address this issue by presenting a novel algorithm named FHM+ for mining HUIs, while considering length constraints. To discover HUIs efficiently with length constraints, FHM+ introduces the concept of Length Upper-Bound Reduction (LUR), and two novel upper-bounds on the utility of itemsets. An extensive experimental evaluation shows that length constraints are effective at reducing the number of patterns, and the novel upper-bounds can greatly decrease the execution time, and memory usage for HUI mining.

Keywords: pattern mining, high-utility itemsets, length constraints

1 Introduction

High-Utility Itemset Mining (HUIM) [2, 13, 5–11] is a popular data mining task. It consists of enumerating all high-utility itemsets (HUIs), i.e. groups of items (itemsets) having a high utility (e.g. yielding a high profit) in customer transaction databases. HUIM is a generalization of the problem of Frequent Itemset Mining (FIM) [1], where items can appear more than once in each transaction and where each item has a weight (e.g. unit profit). HUIM is widely viewed as more difficult than FIM because the utility measure used in HUIM is neither anti-monotonic nor monotonic, i.e. a high utility itemset may have supersets or subsets having lower, equal or higher utilities [2]. HUIM has a wide range of applications [9, 11]. However, an important issue of traditional HUIM algorithms is that they tend to find itemsets containing many items, as they are more likely to have a high utility. This is an issue because itemsets containing many items are generally less useful than itemsets containing fewer items. The reason is that itemsets containing many items generally represent situations that are more specific, and thus rare. For example, consider a retail manager that has found two high-utility itemsets $\{mapleSyrup, pancake\}$, and $\{mapleSyrup, pancake, orange, cheese, cereal\}$ in a customer transaction database. If the retail manager wants to increase the overall profit of his retail store, it will be more effective to promote the first itemset than the second one, as the former contains only two items and may be quite common, while the second contains five items and is rarer. To provide HUIs that are more useful to users while filtering those that may be less useful, it is thus desirable to incorporate the concept of length constraints in HUIM. To our knowledge, no HUIM algorithms offer this feature. To incorporate length constraints in HUIM, a naive approach would be to discover all HUIs using a traditional HUIM algorithm, and then to apply the constraints as a post-processing step. But this approach would be inefficient, as the algorithm would not take advantage of the constraints to prune the search space. Hence, it is desirable to push the constraints as deep as possible in the mining process to improve the performance of the mining task. In frequent pattern mining, length constraints have been previously used such as the maximum length constraint [4]. The key idea of algorithms using a maximum length constraint is that since itemsets are generated by recursively appending items to itemsets, no item should be appended to an itemset containing the maximum number of items. Although this approach can prune the search space using length constraints, there is a need to find novel ways of reducing the search space using length constraints, to further improve the performance of algorithms. In this paper, we address these issues by presenting a novel algorithm named FHM+ (Fast High-utility itemset Mining+) for discovering HUIs with length constraints. It extends the state-of-the-art FHM algorithm for HUIM with a novel concept named Length Upper-bound Reduction (LUR), to reduce the upper-bounds on the utility of itemsets using length constraints, and thus prune the search space. An extensive experimental evaluation shows that the proposed algorithm can be much faster than the state-of-the-art FHM algorithm, and greatly reduce the number of patterns presented to the user. Moreover, results show that the LUR concept greatly improves the algorithm's efficiency. This is an interesting result as the LUR concept introduced in this paper is quite general, and thus could be integrated in other utility pattern mining algorithms. The rest of this paper is organized as follows. Section 2, 3, 4, and 5 respectively present the problem of HUIM and related work, the proposed FHM+ algorithm, the experimental evaluation, and the conclusion.

2 Problem definition and related work

The problem of high-utility itemset mining is defined as follows. Let there be a set of items (symbols) I . A *transaction database* is a set of transactions $D = \{T_1, T_2, ..., T_n\}$ such that for each transaction T_c, $T_c \subseteq I$ and T_c has a unique identifier c called its Tid. Each item $i \in I$ is associated with a positive number $p(i)$, called its external utility (e.g. representing the unit profit of this item). 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. representing the purchase quantity of item i in transaction T_c). For instance, consider the database of Fig. 1, which will be used as running example. It contains five transactions ($T_1, T_2 ... T_5$). For example, transaction T_4 indicates that items a, c, e and g appear in this transaction with an internal utility of respectively 2, 6, 2 and 5. Fig. 2 indicates that the external utilities of these items are respectively 5, 1, 3 and 1. The utility of an

Table 1: A transaction database

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

Table 2: External utility values

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

item i in a transaction T_c is defined as $u(i, T_c) = p(i) \times q(i, T_c)$. The utility of an itemset X (a group of items $X \subseteq I$) in a transaction T_c is defined as $u(X, T_c) = \sum_{i \in X} u(i, T_c)$. 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. The *problem of high-utility itemset mining* is to discover all high-utility itemsets. An itemset X is a *high-utility itemset* if its utility $u(X)$ is no less than a user-specified minimum utility threshold *minutil* given by the user. For example, the utility of item a in T_4 is $u(a, T_4) = 5 \times 2 = 10$. The utility of the itemset $\{a, c\}$ in T_4 is $u(\{a, c\}, T_4) = u(a, T_4) + u(c, T_4) = 5 \times 2 + 1 \times 6 = 16$. The utility of the itemset $\{a, c\}$ is $u(\{a, c\}) = u(a) + u(c) = u(a, T_1) + u(a, T_3) + u(a, T_4) + u(c, T_1) + u(c, T_3) + u(c, T_4) = 5+5+10+1+1+6 = 28$. If $minutil = 30$, the complete set of HUIs is $\{a, c, e\} : 31$, $\{a, b, c, d, e, f\} : 30$, $\{b, c, d\} : 34$, $\{b, c, d, e\} : 40$, $\{b, c, e\} : 37$, $\{b, d\} : 30$, $\{b, d, e\} : 36$, and $\{b, e\} : 31$, where each HUI is annotated with its utility. Several HUIM algorithms have been proposed. They can generally be categorized as one-phase or two-phase algorithms [2]. Two-phase algorithms such as Two-Phase [10], BAHUI [8], PB [5], and UPGrowth+ [11] operate in two phases. In the first phase, they identify itemsets that may be high-utility itemsets by considering an upper-bound on the utility of itemsets called the *Transaction-Weighted Utilization (TWU)* [10]. Then, in the second phase, they scan the database to calculate the exact utility of all candidates found in the first phase and filter those having a low utility. The TWU measure and its pruning property are defined as follows. The *transaction utility* (TU) of a transaction T_c is the sum of the utility of all the items in T_c. i.e. $TU(T_c) = \sum_{x \in T_c} u(x, T_c)$. The *transaction-weighted utilization* (TWU) of an itemset X is defined as the sum of the transaction utility of transactions containing X, i.e. $TWU(X) = \sum_{T_c \in g(X)} TU(T_c)$. For example, The TUs of T_1, T_2, T_3, T_4 and

T_5 are respectively 30, 20, 8, 27 and 11. The TWU of single items a, b, c, d, e, f and g are respectively 65, 61, 96, 58, 88, 30 and 38. $TWU(\{c,d\}) = TU(T_1) + TU(T_2) + TU(T_3) = 30 + 20 + 8 = 58$. Because the TWU measure is anti-monotonic, it can be used to prune the search space.

Property 1 (Pruning search space using the TWU). Let X be an itemset, if $TWU(X) < minutil$, then X and its supersets are low utility. [10]

A drawback of two-phase algorithms is that they generate a huge number of candidates in the first phase. To address this issue, one-phase algorithms were proposed such as FHM [2], HUI-Miner [9], and EFIM [13], which discover HUIs directly using a single phase. To our knowledge, the fastest HUIM algorithm is EFIM, which was shown to outperform FHM, which was shown to be up to 6 times faster than HUI-Miner [2]. The FHM and HUI-Miner algorithms use the concept of remaining utility upper-bound to prune the search space, which is defined as follows. Let $\succ$ be any total order on items from I (e.g. lexicographical order). The *remaining utility of an itemset X in a transaction T_c* is defined as $ru(X) = \sum_{i \in T_c \wedge i \succ x \forall x \in X} u(i, T_c)$. The *remaining utility of an itemset X in a database* is defined as $reu(X) = \sum_{T_c \in g(X)} ru(X, T_c)$. For example, assume that $\succ$ is the alphabetical order. The remaining utility of itemset $\{a, d\}$ in the database is 3, when assuming the alphabetical order. FHM and HUI-Miner are depth-first search algorithms. The FHM algorithm associates a structure named *utility-list* to each itemset [2, 9]. Utility-lists allow calculating the utility of any itemset by making join operations with utility-lists of shorter patterns. Utility-lists are defined as follows. The *utility-list* $ul(X)$ 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 $\sum_{i \in T_{tid} \wedge i \succ x \forall x \in X} u(i, T_{tid})$. The utility-list of $\{a\}$ is $\{(T_1, 5, 25), (T_3, 5, 3), (T_4, 10, 17)\}$. The utility-list of $\{d\}$ is $\{(T_1, 6, 3), (T_2, 6, 3), (T_3, 2, 0)\}$. The utility-list of $\{a, d\}$ is $\{(T_1, 11, 8), (T_3, 7, 0)\}$. The FHM algorithm scans the database once to create the utility-lists of itemsets containing a single item. Then, the utility-lists of larger itemsets are constructed by joining the utility-lists of smaller itemsets. 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 pair 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 $ex.tid = ey.tid = ep.tid$. The utility-list structure allows to calculate the utility-list of itemsets and prune the search space as follows.

Property 2 (Calculating the 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 3 (Pruning search space using a utility-list). 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*, X and its extensions are low utility [9].

Although much work has been done on HUIM, a key problem of current HUIM algorithms is that they tend to find a huge amount of itemsets containing many items. As explained in the introduction, these itemsets may be less useful for users, as they generally represent specific and rare cases. To let users find itemsets that are more useful, we define the problem of mining high-utility itemsets with length constraints as follows.

Definition 1 (High-utility itemset mining with length constraints). *Let minutil, minlength, and maxlength be parameters set by the user. The problem of mining high-utility itemsets with length constraints is to find all itemsets having a utility no less than minutil and containing at least minlength items, and at most maxlength items.*

Example 1. If $minutil = 30$, $minlength = 1$ and $maxlength = 3$, the set of HUIs is: $\{a, c, e\}$, $\{b, c, d\}$, $\{b, c, e\}$, $\{b, d\}$, $\{b, d, e\}$, and $\{b, e\}$.

3 The FHM+ algorithm

This section presents the proposed FHM+ algorithm for efficiently mining HUIs with length constraints. FHM+ extends the state-of-the-art FHM [2] algorithm with novel techniques for pruning the search space using length constraints.

3.1 Length Upper-bound Reduction

As previously mentioned, FHM performs a depth-first search to discover HUIs. To enforce the length constraints in FHM, a simple solution is to modify FHM to not extend an itemset with an item if its number of items is equal to *maxlength*, and to check if the *minlength* constraints is respected for any HUI found by FHM. This approach would find all HUIs when considering length constraints. However, a drawback of this solution is that it does not reduce upper-bounds on the utilities of itemsets to prune the search space. But having tight upper-bounds is crucial in HUIM for pruning the search space efficiently [2, 9, 11]. To address this issue, we next propose a novel concept of *length upper-bound reduction*. It consists of a set of techniques for reducing upper-bounds on the utilities of itemsets using length constraints. This results in two novel tighter upper-bounds on the utility of itemsets called the revised TWU and revised remaining utility. The proposed *revised TWU upper-bound* is defined as follows:

Definition 2 (largest utilities in a transaction). *Let there be a transaction $T_c = \{i_1, i_2, \ldots i_k\}$. The largest utilities in T_c is the set of the maxlength largest values in the set $\{u(i_1, T_c), u(i_2, T_c), \ldots, u(i_k, T_c)\}$, and is denoted as $L(T_c)$.*

Definition 3 (revised Transaction-Weighted Utilization). *Let there be a transaction $T_c = \{i_1, i_2, \dots i_k\}$. The revised transaction utility of T_c is defined as $RTU(T_c) = \sum L(T_c)$, and represents the maximum utility that an itemset respecting the maxlength constraint could have in T_c. The revised TWU of an itemset X is defined as the sum of the revised transaction utilities of transactions where X appears, i.e. $RTWU(X) = \sum_{T_c \in g(X)} RTU(T_c)$.*

For example, the RTU of transactions T_1, T_2, T_3, T_4 and T_5 are respectively 21, 17, 8, 22, and 9. Hence, $RTWU(\{c, d\}) = RTU(T_1) + RTU(T_2) + RTU(T_3) = 21 + 17 + 8 = 48$, which is a tighter upper-bound on the utility of {c,d} and its supersets than the original TWU, which was calculated as 58. The proposed RTWU has the two following important properties.

Property 4 (The revised TWU is a tighter upper-bound than the TWU). Let there be an itemset X. The relationship $RTWU(X) \leq TWU(X)$ holds.

Proof. By definition, $RTU(X) \leq TU(X)$, for any itemset X. Hence, $RTWU(X) = \sum_{T_c \in g(X)} RTU(T_c) \leq TWU(X) = \sum_{T_c \in g(X)} TU(T_c)$.

Property 5 (Pruning the search space using the revised TWU). Let X be an itemset, if $RTWU(X) < minutil$, then X and its supersets are not high-utility itemsets respecting the *maxlength* constraint.

Proof. For any transaction T_c, $RTU(T_c)$ represents the maximum utility that an itemset respecting the *maxlength* constraint could have in T_c. Thus, $RTWU(X)$ is an upper-bound on $u(X)$. Furthermore, it is also a upper-bound on the utilities of supersets of X since those cannot appear in more transactions than X.

The second tighter upper-bound introduced in this paper is called the *revised remaining utility upper-bound.*

Definition 4 (largest utilities in a transaction w.r.t. an itemset). *Let there be a transaction T_c and an itemset X. Let $V(T_c, X) = \{v_1, v_2, \dots v_k\}$ be the set of items occurring in T_c that can extend X, i.e. $V(T_c, X) = \{v \in T_c | v \succ x, \forall x \in X\}$. The maximum number of items that can be appended to X so that the resulting itemset would respect the maxlength constraint is defined as $maxExtend(X) = maxlengh - |X|$, where $|X|$ is the cardinality of X. The largest utilities in transaction T_c with respect to itemset X is the set of the $maxExtend(X)$ largest values in $\{u(v_1, T_c), u(v_2, T_c), \dots, u(v_k, T_c)\}$ and is denoted as $L(T_c, X)$.*

Definition 5 (revised remaining utility). *Let there be a transaction T_c and an itemset X. The revised remaining utility of an itemset X in a transaction T_c is defined as $rru(X, T_c) = \sum L(T_c, X)$, and represents the maximum utility that an extension of X respecting the maxlength constraint could have. The revised remaining utility of an itemset X in a database is defined as $rreu(X) = \sum_{T_c \in g(X)} rru(X, T_c)$.*

For example, the revised remaining utility of itemset $\{a\}$ in the running example is 31, while the remaining utility of $\{a\}$ is 45. This illustrates that the proposed revised remaining utility can be a much tighter upper-bound than the remaining utility upper-bound used in previous work. The proposed revised remaining utility upper-bound has the two following important properties.

Property 6 (The revised remaining utility is a tighter upper-bound than the remaining utility). Let there be an itemset X. The relationship $rreu(X) \leq reu(X)$ holds.

Proof. It can be easily shown that $rru(X) \leq ru(X)$, for any itemset X and transaction T_c. Thus, $rreu(X) = \sum_{T_c \in g(X)} rru(X, T_c) \leq reu(X) = \sum_{T_c \in g(X)} ru(X, T_c)$.

Property 7 (Pruning search space using the revised remaining utility). Let X be an itemset. If the sum of $u(X) + rreu(X)$ is less than $minutil$, X and its extensions are not HUIs respecting the $maxlength$ constraint.

Proof. Since $u(X)$ represents the utility of X, and rreu(X) represents the highest utilities of items that could be appended to X while respecting the $maxlength$ constraint, in transactions where X appears, it follows that the property holds.

We have so far introduced two novel tighter upper-bounds on the utility of itemsets, with the goal of using the *maxlength* constraint for reducing the search space. We next present a novel structure called *revised utility-list* to calculate the revised remaining utility of any itemset efficiently. This structure is a variation of the utility-list structure used in FHM.

Definition 6 (revised utility-list structure). *The* revised utility-list $rul(X)$ *of an itemset X in a database D is a set of tuples such that there is a tuple $(tid, iutil, llist)$ for each transaction T_{tid} containing X. The difference with the utility-lists used in FHM is that the rutil element is replaced by the llist element, which stores the set $L(T_c, X)$.*

For example, consider the running example, with $maxlength = 3$. The revised utility-list of $\{a\}$ is $\{(T_1, 5, \{10, 6\}), (T_3, 5, \{2, 1\}), (T_4, 10, \{6, 6\})\}$, and the revised utility-list of $\{b\}$ is $\{(T_1, 10, \{6, 3\}), (T_2, 8, \{6, 3\}), (T_5, 4, \{3, 2\})\}$. The proposed revised utility-list structure stores the necessary information for pruning an itemset X and its extensions using Property 7.

Property 8 (Pruning search space using the revised utility-list structure). Let X be an itemset. If the sum of the *iutil* values and the *llist* elements in $rul(X)$, is less than $minutil$, X and its extensions are not high-utility itemsets respecting the length constraints.

Although this property is useful, an important question is how to construct the revised utility-list of of any itemset encountered in the search space. This is done as follows. FHM+ initially builds the revised utility-lists of each item by scanning the database. Then, the revised utility-lists of each larger itemset X is obtained by performing a join operation using smaller itemsets. This operation is the same as the join operation of FHM, except that *llist* elements are calculated instead of *rutil* elements, for each tuple in $rul(X)$. Consider two itemsets $P \cup \{x\}$ and $P \cup \{y\}$ such that $x \succ y$. Let there be a tuple *epxy* in the revised utility-list $rul(P \cup \{x, y\})$. Let *ey* be the tuple in $rul(P \cup \{y\})$ such that $ey.tid = epxy.tid$. Recall that $maxExtend(P \cup \{x, y\})$ is the number of items that can be appended to $P \cup \{x, y\}$ to generate an itemset that respects the *maxlength* constraint. The *llist* element of the tuple *epxy* is calculated as the $maxExtend(P \cup \{x, y\})$ largest values in the *llist* element of the tuple *ey*. For example, consider the join of the revised utility-lists of items $\{a\}$ and $\{b\}$ to generate the revised utility-list of itemset $\{a, b\}$, when $maxlength = 3$. The revised utility-list of $\{a, b\}$ contains a single tuple for the transaction T_1. Since $maxExtend(\{a, b\}) = maxlength - |\{a, b\}| = 1$, the *llist* element in the tuple corresponding to T_1 in $rul(\{a, b\})$ is set to the largest utility value in the corresponding *llist* element of $rul(\{b\})$, that is 6. The revised utility-list of $\{a, b\}$ is thus $\{(T_1, 15, \{6\})\}$.

3.2 The proposed algorithm

We next describe the proposed FHM+ algorithm in detail, which relies on the two novel upper-bounds, and revised utility-list structure introduced in the previous subsection. The main procedure of FHM+ (Algorithm 1) takes a transaction database with utility values as input, and the *minutil*, *minlength* and *maxlength* parameters. The algorithm first scans the database once to calculate the RTWU of each item. Then, the algorithm identifies the set I^* of all items having a RTWU no less than *minutil* (other items are ignored since they cannot be part of a high-utility itemset by Property 5). The RTWU values of items are then used to establish a total order $\succ$ on items, which is the order of ascending RTWU values (similarly to the TWU ascending order used in FHM). A database scan is then performed. During this database scan, items in transactions are reordered according to the total order $\succ$, the revised utility-list of each item $i \in I^*$ is built and a structure named EUCS (Estimated Utility Co-Occurrence Structure) is built [2]. This latter structure is defined as a set of triples of the form $(a, b, c) \in I^* \times I^* \times \mathbb{R}$. A triple (a,b,c) indicates that RTWU($\{a, b\}$) $= c$. The EUCS can be implemented as a triangular matrix that stores these triples for all pairs of items. The EUCS is very useful as it stores the RTWU of all pairs of items, an information that will be later used for pruning the search space. Building the EUCS is very fast (it is performed with a single database scan) and occupies a small amount of memory, bounded by $|I^*| \times |I^*|$. The reader is referred to the paper about FHM [2] for more details about the construction of this structure and how it can be implemented efficiently using hash maps. Then, if $minlength \geq 1$, each item having a utility no less than *minutil* according to its revised utility list, is output as a high-utility itemset. 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*, *minlength*, *maxlength*, and the EUCS structure. This exploration is performed only if the user wants to find itemsets containing more than one item.

```
Algorithm 1: The FHM+ algorithm
  input  : $D$: a transaction database, $minutil, minlength, maxlength$:
           user-specified parameters
  output: the set of high-utility itemsets
1 Scan $D$ once to calculate the RTWU of single items;
2 $I^* \leftarrow$ each item $i$ such that RTWU($i$) $\geq minutil$;
3 Let $\succ$ be the total order of RTWU ascending values on $I^*$;
4 Scan $D$ to build the revised utility-list of each item $i \in I^*$ and build the $EUCS$
   structure;
5 if $minlength \leq 1$ then output each item $i \in I^*$ such that
   SUM($\{i\}.utilitylist.iutils$) $\geq minutil$;
6 if $maxlength > 1$ then Search ($\emptyset$, $I^*$, $minutil$, $minlength$, $maxlength$,
   $EUCS$);
```

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*, *minlength*, *maxlength*, and (4) the EUCS. The search procedure operates as follows. For each extension Px of P, if the sum of *iutil* and *llist* values in the revised utility-list of Px are no less than *minutil*, it means that Px and its extensions should be explored (Property 7). 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 revised utility-list of Pxy is then constructed by calling the *Construct* procedure to join the utility-lists of P, Px and Py. This latter procedure performs the steps described in the previous subsection for constructing a revised utility-list. Then, if Pxy respects the length constraints, and the sum of the *iutil* values of the utility-list of Px is no less than *minutil*, then Pxy is a high-utility itemset, and it is output (cf. Property 2). Then, if the length of Pxy is less than *maxlength*, 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 properties 5 and 7, it can be easily seen based on Property 1, 2 and 3 that this procedure is correct and complete to discover all high-utility itemsets, while considering the length constraints.

4 Experimental Study

We performed an experiment to assess the performance of FHM+. The experiment was 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 the proposed FHM+ algorithm with the state-of-the-art FHM algorithm for mining HUIs. All memory measurements were done using the Java API. The experiment was carried on three real-life datasets commonly used in the HUIM literature: *chainstore*, *retail*, and *mushroom*. These datasets have varied charac-

Algorithm 2: The *Search* procedure

```
input  : P: an itemset, ExtensionsOfP: a set of extensions of P, minutil,
         minlength, maxlength: user-specified parameters, EUCS: the EUCS
output: the set of high-utility itemsets
1  foreach itemset $Px \in ExtensionsOfP$ do
2      if SUM(Px.utilitylist.iutils)+SUM(Px.utilitylist.llist) $\geq$ minutil then
3          $ExtensionsOfPx \leftarrow \emptyset$;
4          foreach itemset $Py \in ExtensionsOfP$ such that $y \succ x$ do
5              if $\exists (x, y, c) \in EUCS$ such that $c \geq minutil$ then
6                  $Pxy \leftarrow Px \cup Py$;
7                  $Pxy.utilitylist \leftarrow$ Construct $(P, Px, Py)$;
8                  $ExtensionsOfPx \leftarrow ExtensionsOfPx \cup Pxy$;
9                  if SUM(Pxy.utilitylist.iutils) $\geq$ minutil and
                    $minlength \leq |Pxy| \leq maxlength$ then output $Px$;
10             end
11         end
12         if $|Pxy| < maxlength$ then Search ($Px$, $ExtensionsOfPx$, $minutil$);
13     end
14 end
```

teristics and represent the main types of data typically encountered in real-life scenarios (dense, sparse, and long transactions). Let $|I|$, $|D|$ and A represents the number of transactions, distinct items and average transaction length. *chainstore* is a sparse dataset ($|I|$ = 46,086 $|D|$ = 1,112,949, A = 7.2). *retail* is a sparse dataset with many different items ($|I|$ = 16,470, $|D|$ = 88,162, A = 10,30). *mushroom* is a dense dataset ($|I|$ = 119, $|D|$ = 88,162, A = 23). *chainstore* contains real external and internal utility values. For the other datasets, external utilities for items are generated between 1 and 1,000 by using a log-normal distribution and quantities of items are generated randomly between 1 and 5, as the settings of [2, 9, 11]. The source code of all algorithms and datasets can be downloaded at `http://goo.gl/Qr8diZ`. In the experiment, FHM+ was run with five different *maxlength* threshold values (1, 2, 3, 4, 5), and the *minlength* threshold was set to 1 as it has no influence on efficiency. Algorithms were first run on each dataset, while decreasing the *minutil* threshold until they became too long to execute, ran out of memory or a clear trend was observed. Fig. 1 compares the performance of the algorithms in terms of execution time and pattern count.

It can be first observed that using the *maxlength* constraint can greatly speed up the discovery of HUIs. Depending on how the *maxlength* parameter is set, on the *chainstore*, *retail*, and *mushroom* datasets, FHM+ is respectively from 3 to 10 times , 2 to 17 times, and from 15 to 1400 times faster than FHM. It can also be observed that the number of patterns can be greatly reduced by using length constraints. On the *chainstore*, *retail*, and *mushroom* datasets, the number of patterns found by FHM+ was up to 0.5 times, 13 times, and 2,700 times smaller than the number found by FHM. But note that on the *retail* datasets, no result

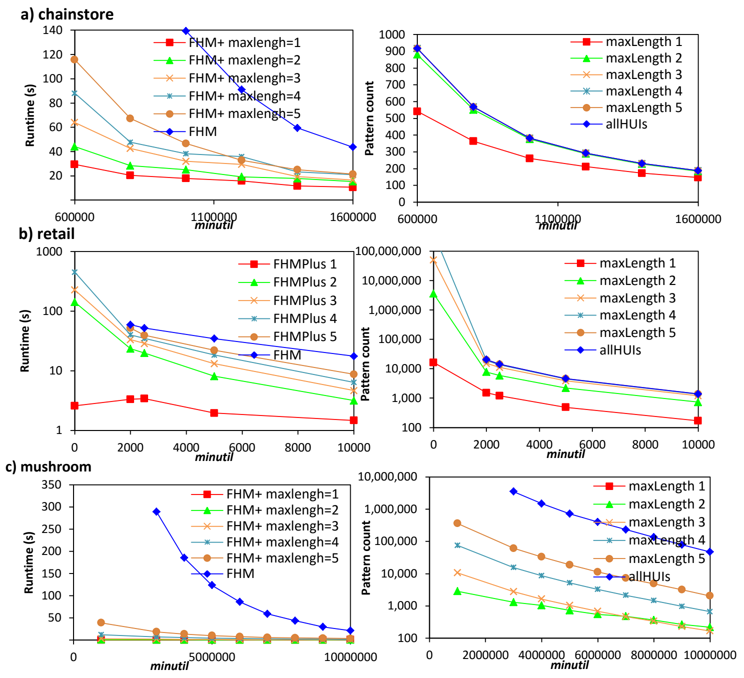

Fig. 1: Execution times and pattern count comparison

was obtained for FHM when *minutil* = *1*, because it had to be stopped after generating more than 20 GB of patterns, while FHM+ was still able to run, even with $maxlength = 5$. Thus, a benefit of using constraints is that it allows the algorithms to run for smaller *minutil* values. Memory consumption was also compared (detailed results are not shown due to space limitations). On the *chainstore*, *retail*, and *mushroom* datasets, FHM+ used from 5% to 50%, 5% to 50%, and 25% to 50% less memory than FHM. This is due to the ability to prune a larger part of the search space using the proposed upper-bounds. Lastly, the efficiency of the proposed LUR concept introduced in this paper was evaluated (detailed results not shown due to space limitations). It was found that on the *chainstore*, *retail*, and *mushroom* datasets, the proposed upper-bound reduced the execution time by up to 4 times, 2 times, and 2 times.

5 Conclusion

This paper presented an efficient algorithm named FHM+ to efficiently discover high-utility itemsets while considering length constraints. The proposed algorithm integrates a novel concept called Length Upper-Bound Reduction (LUR) to reduce the search space using length constraints. In particular, two novel upper-bounds named revised TWU and revised remaining utility were presented, and a novel data structure called revised utility-list. An extensive experimental evaluation shows that LUR is effective at reducing the number of patterns, and can greatly decrease the execution time and memory requirements for HUI mining. This is very interesting for the end user since a huge amount of long HUIs, representing rare cases are filtered and not presented to the user. The source code of algorithms and datasets can be downloaded as part of the SPMF open source data mining library [3] at `http://www.philippe-fournier-viger.com/spmf/`. In future work, the concept of LUR could be incorporated in other utility mining problems such as high-utility sequential rule mining [12]. Moreover, we intend to apply the concept of LUR in the EFIM algorithm [13], which was recently shown to outperform FHM for high-utility itemset mining.

References

1. Agrawal, R., Srikant, R.: Fast algorithms for mining association rules in large databases. In: Proc. Int. Conf. Very Large Databases, pp. 487–499, (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: Proc. 21st Intern. Symp. on Methodologies for Intell. Syst., pp. 83–92 (2014)
3. Fournier-Viger, P., Gomariz, A., Gueniche, T., Soltani, A., Wu., C., Tseng, V. S.: SPMF: a Java Open-Source Pattern Mining Library. Journal of Machine Learning Research (JMLR), 15, pp. 3389-3393 (2014)
4. Pei, J., Han, J.: Constrained frequent pattern mining: a pattern-growth view. ACM SIGKDD Explorations Newsletter, 4(1), 31-39 (2012)
5. Lan, G. C., Hong, T. P., Tseng, V. S.: An efficient projection-based indexing approach for mining high utility itemsets. Knowl. and Inform. Syst. 38(1), 85–107 (2014)
6. Krishnamoorthy, S.: Pruning strategies for mining high utility itemsets. Expert Systems with Applications, 42(5), 2371-2381 (2015)
7. Lin, J. C. W., Gan, W., Hong, T. P., Pan, J. S.: Incrementally Updating High-Utility Itemsets with Transaction Insertion. Proc. 10th Intern. Conf. on Advanced Data Mining and Applications, pp. 44–56 (2014)
8. Song, W., Liu, Y., Li, J.: BAHUI: Fast and memory efficient mining of high utility itemsets based on bitmap. Intern. Journal of Data Warehousing and Mining. 10(1), 1–15 (2014)
9. Liu, M., Qu, J.: Mining high utility itemsets without candidate generation. In: Proc. 22nd ACM Intern. Conf. Info. and Know. Management, pp. 55–64 (2012)
10. Liu, Y., Liao, W., Choudhary, A.: A two-phase algorithm for fast discovery of high utility itemsets. In: Proc. 9th Pacific-Asia Conf. on Knowl. Discovery and Data Mining, pp. 689–695 (2005)
11. 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)
12. Zida, S., Fournier-Viger, P., Wu, C.-W., Lin, J. C. W., Tseng, V.S.: Efficient mining of high utility sequential rules. in: Proc. 11th Intern. Conf. Machine Learning and Data Mining, pp. 1–15 (2015)
13. Zida, S., Fournier-Viger, P., Lin, J. C.-W., Wu, C.-W., Tseng, V.S.: EFIM: A Highly Efficient Algorithm for High-Utility Itemset Mining. Proc. 14th Mexican Intern. Conf. on Artificial Intelligence, pp. 530–546.