# FHM: Faster High-Utility Itemset Mining using Estimated Utility Co-occurrence Pruning

Philippe Fournier-Viger[1], Cheng-Wei Wu[2], Souleymane Zida[1], Vincent S. Tseng[2]

[1] Dept. of Computer Science, University of Moncton, Canada
[2] Dept. of Computer Science and Information Engineering, National Cheng Kung University, Taiwan
philippe.fournier-viger@umoncton.ca, silvemoonfox@gmail.com, esz2233@umoncton.ca, tseng@mail.ncku.edu.tw

**Abstract.** High utility itemset mining is a challenging task in frequent pattern mining, which has wide applications. The state-of-the-art algorithm is HUI-Miner. It adopts a vertical representation and performs a depth-first search to discover patterns and calculate their utility without performing costly database scans. Although, this approach is effective, mining high-utility itemsets remains computationally expensive because HUI-Miner has to perform a costly join operation for each pattern that is generated by its search procedure. In this paper, we address this issue by proposing a novel strategy based on the analysis of item co-occurrences to reduce the number of join operations that need to be performed. An extensive experimental study with four real-life datasets shows that the resulting algorithm named FHM (Fast High-Utility Miner) reduces the number of join operations by up to 95 % and is up to six times faster than the state-of-the-art algorithm HUI-Miner.

**Keywords:** frequent pattern mining, high-utility itemset mining, co-occurrence pruning, transaction database

## 1 Introduction

*Frequent Itemset Mining* (FIM) [1] 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 [1]. 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 mining 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 itemsets having a high utility (e.g. 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 [2, 7, 10]. HUIM has also inspired several important data mining tasks such as high-utility sequential pattern mining [11] and high-utility stream mining [9].

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 [1]. 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 [2, 6–8, 10]. A popular approach to HUIM is to discover high-utility itemsets in two phases using the Transaction-Weigthed-Downward closure model [8, 2, 10]. This approach is adopted by algorithms such as Two-Phase [8], IHUP [2] and UPGrowth [10]. These algorithms first generate a set of candidate high-utility itemsets by overestimating their utility in Phase 1. Then, in Phase 2, the algorithms perform a database scan to calculate the exact utility of candidates and filter low-utility itemsets. Recently, a more efficient approach was proposed in the HUI-Miner algorithm [7] to mine high-utility itemsets directly using a single phase. HUI-Miner was shown to outperform previous algorithms and is thus the current best algorithm for HUIM [7]. However, the task of high-utility itemset mining remains very costly in terms of execution time. Therefore, it remains an important challenge to design more efficient algorithms for this task.

In this paper, we address this challenge. Our proposal is based on the observation that although HUI-Miner performs a single phase and thus do not generate candidates as per the definition of the two-phase model, HUI-Miner explores the search space of itemsets by generating itemsets and a costly join operation has to be performed to evaluate the utility of each itemset. To reduce the number of joins that are performed, we propose a novel pruning strategy named EUCP (Estimated Utility Cooccurrence Pruning) that can prune itemsets without having to perform joins. This strategy is easy to implement and very effective. We name the proposed algorithm incorporating this strategy FHM (Fast High-utility Miner). We compare the performance of FHM and HUI-Miner on four real-life datasets. Results show that FHM performs up to 95 % less join operations than HUI-Miner and is up to six times faster than HUI-Miner

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

**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 $u(X, T_c) = \sum_{i \in X} u(i, T_c)$.

*Example 2.* The utility of item $a$ in $T_2$ is $u(a, T_2) = 5 \times 2 = 10$. The utility of the itemset $\{a, c\}$ in $T_2$ is $u(\{a, c\}, T_2) = u(a, T_2) + u(c, T_2) = 5 \times 2 + 1 \times 6 = 16$.

**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 $\{a, c\}$ is $u(\{a, c\}) = u(a) + u(c) = u(a, T_1) + u(a, T_2) + u(a, T_3) + u(c, T_1) + u(c, T_2) + u(c, T_3) = 5 + 10 + 5 + 1 + 6 + 1 = 28$.

**Definition 4 (problem definition).** 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. Otherwise, $X$ is a *low-utility itemset.*

*Example 4.* If $minutil = 30$, the high-utility itemsets in the database of our running example are $\{b, d\}$, $\{a, c, e\}$, $\{b, c, d\}$, $\{b, c, e\}$, $\{b, d, e\}$, $\{b, c, d, e\}$ with respectively a utility of 30, 31, 34, 31, 36, 40 and 30.

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) [2, 8, 10], which is anti-monotonic. 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. $TU(T_c) = \sum_{x \in T_c} u(x, T_c)$.

*Example 5.* Fig. 2 (left) shows the TU of transactions $T_1, T_2, T_3, T_4, T_5$ from our running example.

**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 6.* Fig. 2 (center) shows the TWU of single items $a$, $b$,$c$, $d$, $e$, $f$, $g$. Consider item $a$. $TWU(A) = TU(T_1) + TU(T_2) + TU(T_3) = 8 + 27 + 30 = 65$

The TWU measure has three important properties that are used to prune the search space.

*Property 1 (overestimation).* The TWU of an itemset $X$ is higher than or equal to its utility, i.e. $TWU(X) \geq u(X)$ [8].

*Property 2 (antimonotonicity).* The TWU measure is anti-monotonic. Let $X$ and $Y$ be two itemsets. If $X \subset Y$, then $TWU(X) \geq TWU(Y)$ [8].

*Property 3 (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. Proof. This directly follows from Property 1 and Property 2.

Algorithms such as Two-Phase [8], IHUP [2] and UPGrowth [10] utilizes the aforementionned properties 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. Recently, an alternative approach was proposed in the HUI-Miner algorithm [7] to mine high-utility itemsets directly using a single phase. HUI-Miner was shown to outperform previous algorithms and is thus the current best algorithm for HUIM [7]. HUI-Miner utilizes a depth-first search to explore the search space of itemsets. HUI-Miner associate a structure named *utility-list* [7] 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 7 (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 $\sum_{i \in T_{tid} \wedge i \notin X} U(i, T_{tid})$.

*Example 7.* The utility-list of $\{a\}$ is $\{(T_1, 5, 3)(T_2, 10, 17)(T_3, 5, 25)\}$. The utility-list of $\{e\}$ is $\{(T_2, 6, 5)(T_3, 3, 5)(T_4, 3, 0)\}$. The utility-list of $\{a, e\}$ is $\{(T_2, 16, 5),$ $(T_3, 8, 5)\}$.

To discover high-utility itemsets, HUI-Miner perform a single database scan to create utility-lists of patterns containing single items. Then, larger patterns are obtained by performing the join operation of utility-lists of smaller patterns. Pruning the search space is done using the two following properties.

*Property 4 (sum of iutils).* Let $X$ be an itemset. If the sum of *iutil* values in the utility-list of $x$ is higher than or equal to *minutil*, then $X$ is a high-utility itemset. Otherwise, it is a low-utility itemset [7].

*Property 5 (sum of iutils and rutils).* 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 $y \succ i$ for all item $i$ in $X$. If the sum of *iutil* and *rutil* values in the utility-list of $x$ is less than *minutil*, all extensions of $X$ and their transitive extensions are low-utility itemsets [7].

HUI-Miner is a very efficient algorithm. However, a drawback is that the join operation to calculate the utility-list of an itemset is very costly. In the next section, we introduce our novel algorithm, which improves upon HUI-Miner by being able to eliminate low-utility itemsets without performing join operations.

| **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)

| **TID** | **TU** |
|---|---|
| $T_1$ | 8 |
| $T_2$ | 27 |
| $T_3$ | 30 |
| $T_4$ | 20 |
| $T_5$ | 11 |

| **Item** | **TWU** |
|---|---|
| a | 65 |
| b | 61 |
| c | 96 |
| d | 58 |
| e | 88 |
| f | 30 |
| g | 38 |

| **Item** | a | b | c | d | e | f |
|---|---|---|---|---|---|---|
| b | 30 | | | | | |
| c | 65 | 61 | | | | |
| d | 38 | 50 | 58 | | | |
| e | 57 | 61 | 77 | 50 | | |
| f | 30 | 30 | 30 | 30 | 30 | |
| g | 27 | 38 | 38 | 0 | 38 | 0 |

**Fig. 2.** Transaction utilities (left), TWU values (center) and EUCS (right)

## 3 The FHM algorithm

In this section, we present our proposal, the FHM algorithm. 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 itemsets 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 [7]). 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 our novel structure named EUCS (Estimated Utility Co-Occurrence Structure) is built. 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 TWU($\{a, b\}$) $= c$. The EUCS can be implemented as a triangular matrix as shown in Fig. 2 (right) or as a hashmap of hashmaps where only tuples of the form $(a, b, c)$ such that $c \neq 0$ are kept. In our implementation, we have used this latter representation to be more memory efficient because we have observed that few items co-occurs with other items. 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 (cf. section 5). 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 structure.

**Algorithm 1:** The FHM 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($i$) $< minutil$;
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 ($\emptyset$, $I^*$, $minutil$, $EUCS$);
```

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 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. ). 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 HUI-Miner 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 HUI-Miner [7] 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 explore the search space of itemsets by appending single items and it only prunes the search space based on Property 5. It can be easily seen based on Property 4 and 5 that this procedure is correct and complete to discover all high-utility itemsets.

**Co-occurrence-based Pruning.** The main novelty in FHM is a novel pruning mechanism named EUCP (Estimated Utility Co-occurrence Pruning), which relies on a new structure, the EUCS. EUCP is based on the observation that one of the most costly operation in HUI-Miner is the join operation. EUCP is a pruning strategy to directly eliminate a low-utility extension $Pxy$ and all its transitive extensions without constructing their utility-list. This is done on line 8 of the *Search* procedure. The pruning condition is that if there is no tuple $(x, y, c)$ in EUCS such that $c \geq minutil$, then $Pxy$ and all its supersets are low-utility itemsets and do not need to be explored.

This strategy is correct (only prune low-utility itemsets). The proof is that by Property 3, if an itemset $X$ contains another itemset $Y$ such that $TWU(Y) < minutil$, then $X$ and its supersets are low-utility itemsets.

An important question about the EUCP strategy is: should we not only check the condition for $x, y$ in each call to *Search* but also check the condition for all pairs of distinct items $a, b \in Pxy$? The answer is no because the *Search* procedure is recursive and therefore all other pairs of items in $Pxy$ have already been checked in previous recursions of the *Search* procedure leading to $Pxy$. For example, consider an itemset $Z = \{a_1, a_2, a_3, a_4\}$. To generate this itemset, the search procedure had to combine $\{a_1, a_2\, a_3\}$ and $\{a_1, a_2, a_4\}$, obtained by combining $\{a_1, a_2\}$ and $\{a_1, a_3\}$, and $\{a_1, a_2\}$ and $\{a_1, a_4\}$, obtained by combining single items. It can be easily observed that when generating $Z$ all pairs of items in $Z$ have been checked by EUCP except $\{a_3, a_4\}$.

## 4 Experimental Study

We performed 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 FHM with the state-of-the-art algorithm HUI-Miner for high-utility itemset mining. All memory measurements were done using the Java API. Experiments were carried on four real-life datasets having varied characteristics. The *Chainstore* dataset contains 1,112,949 transactions with 46,086 distinct items and an average transaction length of 7.26 items. The *BMS* dataset contains 59,601 transactions with 497 distinct items and an average transaction length of 4.85 items. The *Kosarak* dataset contains 990,000 transactions with 41,270 distinct items and an average transaction length of 8.09 items. The *Retail* dataset contains 88,162 transactions with 16,470 distinct items and an average transaction length of 10,30 items. The Chainstore dataset already contain unit profit

**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 $Px \in ExtensionsOfP$ do
2      if $SUM(Px.utilitylist.iutils) \geq minutil$ then
3          output $Px$;
4      end
5      if $SUM(Px.utilitylist.iutils)+SUM(Px.utilitylist.rutils) \geq minutil$ then
6          $ExtensionsOfPx \leftarrow \emptyset$;
7          foreach itemset $Py \in ExtensionsOfP$ such that $y \succ x$ do
8              if $\exists(x, y, c) \in EUCS$ such that $c \geq minutil)$ then
9                  $Pxy \leftarrow Px \cup Py$;
10                 $Pxy.utilitylist \leftarrow$ Construct $(P, Px, Py)$;
11                 $ExtensionsOfPx \leftarrow ExtensionsOfPx \cup Pxy$;
12             end
13         end
14         Search ($Px$, $ExtensionsOfPx$, $minutil$);
15     end
16 end
```

information and purchase quantities. For 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, 7, 10]. The source code of all algorithms and datasets can be downloaded from `http://goo.gl/hDtdt`.

**Execution time**. We first ran the FHM and HUI-Miner 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, the percentage of candidate pruned by the FHM algorithm and the total size of the EUCS. The comparison of execution times is shown in Fig. 3. For Chainstore, BMS, Kosarak and Retail, FHM was respectively up to 6.12 times faster, 6 times faster, 4.33 times faster and 2.3 times faster than HUI-Miner.

**Pruning effectiveness**. The percentage of candidates pruned by the FHM algorithm was 18% to 91%, 87%, 87 % and 31% to 95% for the Chainstore, BMS, Kosarak and retail datasets. These results show that candidate pruning can be very effective by pruning up to 95 % of candidates. As expected, when more pruning was done, the performance gap between FHM and HUI-Miner became larger.

**Memory overhead**. We also studied the memory overhead of using the EUCS structure. We found that for the Chainstore, BMS, Kosarak and Retail datasets, the memory footprint of EUCS was respectively 10.3 MB, 4.18 MB,

**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  $UtilityListOfPxy \leftarrow \emptyset$;
2  foreach  tuple $ex \in Px.utilitylist$ do
3      if $\exists ey \in Py.utilitylist$ and $ex.tid = exy.tid$ then
4          if $P.utilitylist \neq \emptyset$ then
5              Search element $e \in P.utilitylist$ such that $e.tid = ex.tid.$;
6              $exy \leftarrow (ex.tid, ex.iutil + ey.iutil - e.iutil, ey.rutil)$;
7          end
8          else
9              $exy \leftarrow (ex.tid, ex.iutil + ey.iutil, ey.rutil)$;
10         end
11         $UtilityListOfPxy \leftarrow UtilityListOfPxy \cup \{exy\}$;
12     end
13 end
14 return $UtilityListPxy$;
```

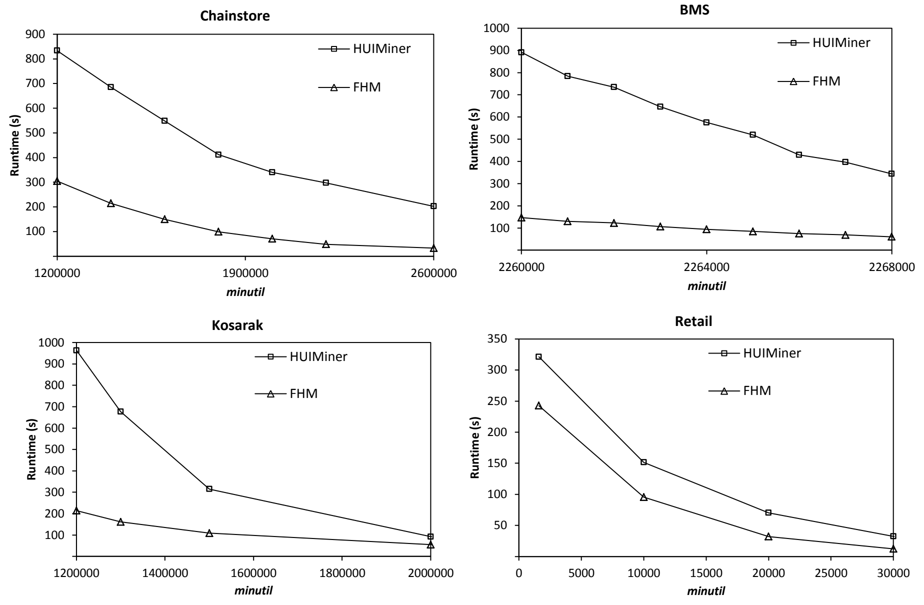

**Fig. 3.** Execution times

1.19 MB and 410 MB. We therefore conclude that the cost of using the EUCP strategy in terms of memory is low.

## 5 Conclusion

In this paper, we have presented a novel algorithm for high-utility itemset mining named FHM (Fast High-Utility Miner). This algorithm integrates a novel strategy named EUCP (Estimated Utility Cooccurrence Pruning) to reduce the number of joins operations when mining high-utility itemsets using the utility-list data structure. We have performed an extensive experimental study on four real-life datasets to compare the performance of FHM with the state-of-the-art algorithm HUI-Miner. Results show that the pruning strategy reduces the search space by up to 95 % and that FHM is up to 6 times faster than HUI-Miner. The source code of all algorithms and datasets used in our experiments can be downloaded from `http://goo.gl/hDtdt`, as part of the SPMF data mining library.

For future work, we are interested in exploring other optimizations for itemset mining, sequential pattern mining [3, 4] and sequential rule mining [5].

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

## 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. 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)
3. 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)
4. 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)
5. Fournier-Viger, P., Nkambou, R., Tseng, V. S.: RuleGrowth: Mining Sequential Rules Common to Several Sequences by Pattern-Growth. In: Proc. ACM 26th Symposium on Applied Computing, pp. 954–959 (2011)
6. 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)
7. Liu, M., Qu, J.:Mining High Utility Itemsets without Candidate Generation. In Proceedings of CIKM12, pp. 55–64 (2012)
8. 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)
9. 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)
10. 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)
11. 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)