Intelligent Data Analysis 21 (2017) 1407–1423
DOI 10.3233/IDA-163180
IOS Press


# Efficiently mining of skyline frequent-utility patterns

Jeng-Shyang Pan[a], Jerry Chun-Wei Lin[b,*], Lu Yang[b], Philippe Fournier-Viger[c] and Tzung-Pei Hong[d,e]
[a]*Fuzhou University of International Studies and Trade, Fuzhou, Fujian, China*
[b]*School of Computer Science and Technology, Harbin Institute of Technology Shenzhen Graduate School, Shenzhen, Guangdong, China*
[c]*School of Natural Sciences and Humanities, Harbin Institute of Technology Shenzhen Graduate School, Shenzhen, Guangdong, China*
[d]*Department of Computer Science and Information Engineering, National University of Kaohsiung, Kaohsiung, Taiwan*
[e]*Department of Computer Science and Engineering, National Sun Yat-sen University, Kaohsiung, Taiwan*

**Abstract.** Frequent itemset mining (FIM) is one of the most common data mining techniques, which is based on the analysis of the occurrence frequencies of items in transactions. However, it is inapplicable in real-life situations since customers may purchase several units of the same item and all items may not have the same unit profits. High-utility itemset mining (HUIM) was designed to consider both the quantities and unit profits of items in databases, and has become an emerging and critical research topic in recent decades. The SKYMINE approach was proposed to mine the skyline frequent-utility patterns (SFUPs), by considering both the utility and the occurrence frequencies of items. A SFUP is a non-dominated itemset, where the dominance relationship between itemsets is based on the utility and frequency measures. Mining SFUPs using the SKYMINE algorithm and its (UP)-tree structure requires, however, long execution times. In this paper, we propose a more efficient algorithm named skyline frequency-utility (SFU)-Miner to mine the SFUPs, utilizing the utility-list structure. This latter structure is used to efficiently calculate the actual utilities of itemsets without generating candidates, contrarily to the SKYMINE algorithm and its UP-tree structure. Besides, an array called utility-max (*umax*) is further developed to keep information about the maximal utility for each occurrence frequency, which can be used to greatly reduce the amount of itemsets considered for directly mining the SFUPs. This property can be used to efficiently find the non-dominated itemsets based on the utility and frequency measures. Substantial experiments have been carried out to evaluate the proposed algorithm's performance. Results have shown that SFU-Miner outperforms the state-of-the-art SKYMINE algorithm for SFUP mining in terms of runtime, memory consumption, number of candidates, and scalability.

Keywords: Data mining, umax array, skyline frequent-utility patterns, frequency, utility

## 1. Introduction

Knowledge discovery in databases (KDD) has became an important research topic in recent decades since its techniques can be used to find potential and implicit information in very large databases. Frequent itemset mining (FIM) is a fundamental KDD task, used to identify the set of frequent itemsets

*Corresponding author: Jerry Chun-Wei Lin, Harbin Institute of Technology Shenzhen Graduate School, Shenzhen, Guangdong, China. Email: jerrylin@ieee.org.

(FIs). An itemset is considered as a FI if its occurrence frequency is no less than a minimum support threshold. Many methods have been proposed to discover FIs. They generally belong to one of two categories: level-wise [3,25,28] and pattern-growth [12,13,17] approaches. The most famous level-wise algorithm is called Apriori [3]. It utilizes a generate-and-test mechanism to produce candidate itemsets of different sizes (levels). Apriori scans the database for each level to retrieve the FIs from the generated candidates. These FIs are then combined to generate the candidate itemsets of the next level. This recursive process is repeated and guarantees finding all FIs respecting the minimum support threshold constraint. Then, the discovered FIs can be combined to obtain association rules (ARs) respecting a minimum confidence threshold. The most representative pattern-growth algorithm is called FP-growth [13]. It discovers FIs by recursively exploring a designed FP-tree structure. The FP-tree structure maintains information about the occurrence frequencies of all frequent items. The FP-growth algorithm recursively explores that structure and performs projections to count the support of itemsets. Both level-wise and pattern-growth algorithms treat all items in a database as binary variables. That is, they consider that items may appear or not in transactions, but they do not consider their quantities. Besides, another drawback of FIM and association-rule mining (ARM) algorithms is that they only consider the support of items, but not other measures that are important in real-life such as the prices, unit profits, interestingness, and weights of items. In real-life situations, the occurrence frequencies of itemsets are generally not representative of their importance to the user. For instance, consider the use of FIM to analyze customer transactions for the purpose of refining marketing strategies in terms of sales and discounts. Albeit it may be found that the purchase of bread is very common, the sale of diamonds may be much less frequent but may still be globally more profitable than the sale of bread. The reason is that the profit yield by the sale of one diamond is much higher than the profit obtained by the sale of one bread. For this reason, only considering the occurrence frequencies of items is inadequate to find highly profitable itemsets.

To solve the aforementioned limitations of FIM, several algorithms have been designed for high-utility itemset mining (HUIM) [5,31–33,35,36]. Their purpose is to discover "useful" and "profitable" itemsets in quantitative databases. An itemset is considered as a high-utility itemset (HUI) if its utility is no less than a predefined minimum utility threshold. The utility mining problem was introduced by Chan et al. [7] as a generalization of FIM. Yao et al. [31] then defined the concept of internal and external utilities, which represent the purchase quantities of items in transactions and their unit profits, respectively.

Albeit FIM and HUIM have numerous real-world applications, they both uses only one criterion to assess the interestingness of patterns (either their occurrence frequencies or utilities). But in some cases, both the frequency and utility measures are relevant to users. As a solution, Goyal et al. [11] defined the skyline frequent-utility pattern (SFUP) mining problem and the SKYMINE algorithm to mine these itemsets. The algorithm can be used to find itemsets that are both highly frequent and profitable. An itemset $X$ is called a SFUP if no other itemset in the database has both higher frequency and utility than those of $X$. The SKYMINE algorithm discovers SFUPs by first producing candidates using the UP-tree structure. These candidates are then evaluated by scanning the database to determine their exact utilities and obtain the set of SFUPs. This process can, however, be very time-consuming and requires a large amount of memory since numerous candidates may be generated. To speed up the mining process and reduce memory usage for mining SFUPs, this paper proposes a new algorithm called skyline frequent-utility (SFU)-Miner. The major contributions of this paper are as follows:

1. An efficient utility-list structure is adopted in this paper to efficiently mine the set of SFUPs. The utility-list structure allows finding $k$-itemsets ($k \geqslant 2$) using a simple join operation. This process can be used to reduce th e amount of candidates for mining SFUPs.

2. An *umax* array is further developed to keep information about the maximal utility for each occurrence frequency. This array guarantees that the algorithm is correct and complete to discover all non-dominated itemsets. Hence, the algorithm is highly efficient in terms of runtime and memory usage.
3. Using the designed SFU-Miner algorithm, the user can discover the set of SFUPs without having to set a minimum utility threshold or minimum support threshold, while considering both the occurrence frequencies and utility constraints.

## 2. Related work

In this section, work related to HUIM and the concept of skyline patterns are briefly reviewed.

### *2.1. High-utility itemset mining*

High-utility itemset mining (HUIM) is an extension of FIM, which considers both the quantities and the unit profits of itemsets in databases. Yao et al. [32] proposed the UMining algorithm which utilizes an estimation method to prune the search space. It cannot, however, discover the complete set of HUIs since some itemsets may be incorrectly pruned by its proposed estimation method. To obtain a downward closure (DC) property for mining all HUIs, Liu et al. [21] then developed a two-phase (TWU) algorithm. It mines HUIs by first identifying a set of itemsets called the high transaction-weighted-utilization itemsets (HTWUIs), to maintain a transaction-weighted downward closure (TWDC) property. Based on the TWDC property, unpromising itemsets can be pruned early while still ensuring that all HUIs can be discovered. Although the TWU model mines the complete set of HUIs, numerous candidate HTWUIs are still generated, and multiple database scans are required to mine HUIs in a level-wise way. To solve the above limitations, pattern-growth approaches such as the IHUP-tree [5], HUP-tree [18] and UP-Growth+ [30] algorithms have been designed. The IHUP-tree algorithm maintains a tree structure to interactively mine HUIs. Transactions are sorted in three different orders for evaluation. The HUP-tree algorithm keeps quantity information about items in its tree structure and the HUP-growth algorithm is applied to directly mine HUIs in the compressed HUP-tree structure. The UP-Growth+ algorithm [30] adopts several pruning strategies to speed up the mining process based on the developed utility-pattern (UP)-tree.

Besides pattern-growth approaches, a novel list-based algorithm called HUI-Miner [22] was developed to mine HUIs without candidate generation. It consumes less memory than previous algorithms but can still efficiently retrieve all HUIs. Each tuple of the designed utility-list structure keeps three elements, which respectively are the transaction ids (*tid*), the utility value of an item in a transaction (*iutil*) and the remaining utility value of an item in a transaction (*rutil*). The FHM algorithm [10] was then developed. It relies on the utility-list structure and a triangular matrix to maintain information about the utility of 2-itemsets (pairs of items), and speed up the discovery of HUIs. Many other studies on HUIM have also been recently carried out [10,19,36].

Although algorithms for FIM and HUIM can mine interesting and meaningful information, they only consider one measure for selecting patterns, that is either the occurrence frequency or the utility. Yeh et al. [34] first proposed the two-phase algorithm for mining high utility itemsets with high frequencies. Podpecan et al. [27] then proposed a fast algorithm to mine the frequent-utility itemsets. However, it remains difficult to define an appropriate minimum utility threshold and minimum support threshold

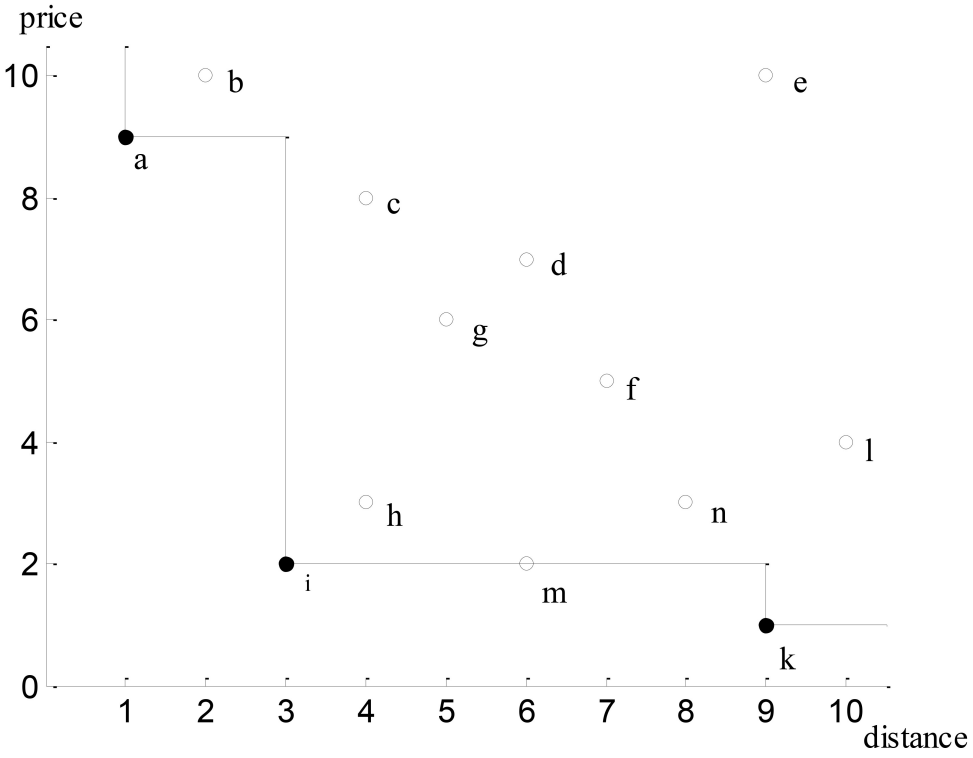


Fig. 1. The skyline of a 2-dimensional dataset.

to retrieve information that is relevant to users. Thus, Goyal et al. proposed an efficient skyline itemset mining (SKYMINE) algorithm [11] to discover the skyline frequent-utility patterns (SFUPs). Using SKYMINE, the user does not need to define thresholds to find the complete set of non-dominated itemsets, by considering both the occurrence frequency and utility factors. The SKYMINE algorithm relies on the UP-tree structure [30] to store the required information for mining SFUPs using a pattern-growth approach. However, the SKYMINE algorithm fails to handle very large databases since its level-wise approach for mining SFUPs often produces a large number of candidates.

### *2.2. Skyline concept*

The concept of skyline has become an important topic in recent years since it can return a set of interesting points or information from a huge data space. It is based on the notion of dominance between tuples when multiple dimensions of a dataset are considered. Given a set of $n$ objects, the skyline of these objects are the objects that are not dominated by any other objects. The domination of an object $Y$ by another object $X$ indicates that $X$ is as good or better for all dimensions, and better than $Y$ for at least one dimension. For example, consider Fig. 1, which depicts the characteristics of various hotels in terms of distance to the beach ($x$ axis) and price ($y$ axis). The skyline points of these hotels are $a$, $i$, and $k$ since no points have higher values than them for these dimensions.

Borzsonyi et al. [6] first considered the concept of skylines in database systems and developed several algorithms based on the use of block nested loops, a divide-and-conquer approach, and an index scanning mechanism. An improved version of block nested loops was presented by Chomicki et al. [9],

Table 1
A quantitative database

| TID | Items with their quantities |
|---|---|
| 1 | $(B, 1)$ $(C, 2)$ $(D, 1)$ |
| 2 | $(A, 4)$ $(B, 1)$ $(C, 3)$ $(D, 1)$ $(E, 1)$ |
| 3 | $(A, 4)$ $(C, 2)$ $(D, 1)$ |
| 4 | $(C, 2)$ $(E, 1)$ |
| 5 | $(A, 5)$ $(B, 2)$ $(D, 1)$ $(E, 2)$ |
| 6 | $(A, 3)$ $(B, 4)$ $(C, 1)$ |
| 7 | $(D, 1)$ |

which employs an ordering of tuples in a window to increase performance. Tan et al. [29] proposed progressive (or on-line) algorithms that can progressively output skyline points without scanning the entire dataset. Kossmann et al. [14] presented a NN algorithm to apply a divide-and-conquer approach based on a nearest-neighbor search optimized using R-trees. The experimental evaluation has shown that NN outperforms previous algorithms in terms of overall performance for datasets having varied characteristics. Another benefit of this approach is the support for online processing. However, NN has also some serious shortcomings such as the needs for duplicate elimination, multiple node visits, and large space requirements. Motivated by this fact, Papadias et al. [24] proposed a progressive algorithm called branch-and-bound skyline (BBS), which utilizes a nearest-neighbor search considering I/O costs, to achieve better result. The discovery of skylines in databases is an active research area [2,8,15,16].

In traditional HUIM or FIM, only one measure is considered to extract patterns from databases. But in real-life, it is beneficial to consider more than one measure. It is thus an interesting topic to apply the skyline concept in high utility mining to consider both the occurrence frequency and utility measures. In this paper, an efficient algorithm called SFU-Miner is proposed based on the utility-list structure to first generate the potential SFUs. This utility-list structure can be used to reduce the combinatorial cost by using a simple join operation to calculate the utilities of itemsets and upper-bounds to reduce the search space. Besides, an utility-max (*umax*) array is designed to efficiently identify SFUPs from the set of potential SFUs. The developed pruning strategy can greatly reduce the search space for mining SFUPs. The preliminaries and the problem statement of skyline frequent-utility pattern mining (SFUPM) are presented next.

## 3. Preliminaries and problem statement

### 3.1. Preliminaries

Let $I = \{i_1, i_2, \ldots, i_m\}$ be a finite set of $m$ distinct items. A quantitative database is a set of transactions $D = \{T_1, T_2, \ldots, T_n\}$, where each transaction $T_q \in D$ $(1 \leqslant q \leqslant n)$ is a subset of $I$ and has a unique identifier $q$, called its $TID$. Besides, in a transaction $T_q$, the purchase quantity of each item $i_j$ (internal utility) is denoted as $q(i_j, T_q)$. A profit table ptable $= \{pr(i_1), pr(i_2), \ldots, pr(i_m)\}$ indicates the profit value (external utility) of each item $i_j$. A set of $k$ distinct items $X = \{i_1, i_2, \ldots, i_k\}$ such that $X \subseteq I$ is said to be a $k$-itemset, where $k$ is the length of the itemset. An itemset $X$ is said to be contained in a transaction $T_q$ if $X \subseteq T_q$. For example, a small quantitative database is shown in Table 1, which will be used as running example. This database has 7 transactions and 5 distinct items, denoted from $(A)$ to $(E)$, respectively. The profit values (external utilities) of items are shown in Table 2.

Table 2
A profit table

| Item | $A$ | $B$ | $C$ | $D$ | $E$ |
|---|---|---|---|---|---|
| Profit | 1 | 2 | 1 | 5 | 4 |

**Definition 1.** The occurrence frequency of an itemset $X$ in a database $D$ is denoted as $f(X)$, where $X$ is a set of items and $f(X)$ is defined as the number of transactions that contain $X$, that is:

$$f(X) = |\{X \subseteq T_q \wedge T_q \in D\}|. \tag{1}$$

For example, the frequency of $(D)$ and $(BD)$ are respectively 5 and 3 since $(D)$ appears five times (in transactions $T_1$, $T_2$, $T_3$, $T_5$ and $T_7$) and $(BD)$ appears 3 times (in transactions $T_1$, $T_2$ and $T_5$).

**Definition 2.** The utility of an item $i_j$ in a transaction $T_q$ is denoted as $u(i_j, T_q)$ and defined as:

$$u(i_j, T_q) = q(i_j, T_q) \times pr(i_j). \tag{2}$$

For example, the utility of the items $(B)$ and $(D)$ in transaction $T_1$ are respectively calculated as $u(B, T_1) = q(B, T_1) \times pr(B) = 1 \times 2$ $(= 2)$ and $u(D, T_1) = q(D, T_1) \times pr(D) = 1 \times 5$ $(= 5)$.

**Definition 3.** The utility of an itemset $X$ in a transaction $T_q$ is denoted as $u(X, T_q)$ and defined as:

$$u(X, T_q) = \sum_{i_j \subseteq X \wedge X \subseteq T_q} u(i_j, T_q). \tag{3}$$

For example, the utilities of the itemsets $(D)$ and $(BD)$ in transaction $T_1$ are calculated as $u(D, T_1) = q(D, T_1) \times pr(D) = 1 \times 5$ $(= 5)$ and $u(BD, T_1) = q(B, T_1) \times pr(B) + q(D, T_1) \times pr(D) = 1 \times 5 + 1 \times 2$ $(= 7)$, respectively.

**Definition 4.** The utility of an itemset $X$ in a database $D$ is denoted as $u(X)$, and defined as:

$$u(X) = \sum_{X \subseteq T_q \wedge T_q \in D} u(X, T_q). \tag{4}$$

For example, the utilities of the itemsets $(D)$ and $(BD)$ in $D$ are calculated as $u(D) = u(D, T_1) + u(D, T_2) + u(D, T_3) + u(D, T_5) + u(D, T_7) = 5 + 5 + 5 + 5 + 5$ $(= 25)$, and $u(BD) = u(BD, T_1) + u(BD, T_2) + u(BD, T_5) = 7 + 7 + 9$ $(= 23)$, respectively.

**Definition 5.** The transaction utility of a transaction $T_q$ is denoted as $tu(T_q)$ and defined as:

$$tu(T_q) = \sum_{i_j \subseteq T_q} u(i_j, T_q). \tag{5}$$

For example, $tu(T_1) = u(B, T_1) + u(C, T_1) + u(D, T_1) = 2 + 2 + 5$ $(= 9)$. The transaction utilities of the other transactions can be calculated in the same way, to otain $tu(T_1)$ $(= 9)$, $tu(T_2)$ $(= 18)$, $tu(T_3)$ $(= 11)$, $tu(T_4)$ $(= 6)$, $tu(T_5)$ $(= 22)$, $tu(T_6)$ $(= 12)$, and $tu(T_7)$ $(= 5)$.

**Definition 6.** The transaction-weighted utility of an itemset $X$ in a database $D$ is denoted as $twu(X)$ and defined as:

$$twu(X) = \sum_{X \subseteq T_q \wedge T_q \in D} tu(T_q). \tag{6}$$

For example, *twu(D)* $= tu(D, T_1) + tu(D, T_2) + tu(D, T_3) + tu(D, T_5) + tu(D, T_7) = 9 + 18 + 11 + 22 + 5$ $(= 65)$.

The above definitions have been used in previous work to determine if an itemset is a FI or HUI. The state-of-the-art algorithms for FIM and HUIM consider either the frequency or the utility measure to select itemsets. The concept of skyline frequent-utility patterns (SFUPs) is based on the following definitions.

**Definition 7.** An itemset $X$ dominates another itemset $Y$ in a database $D$, denoted as $X \succ Y$ iff $f(X) \geqslant f(Y)$ and $u(X) \geqslant u(Y)$.

For example, $(D) \succ (BD)$ since $u(D) > u(BD)$ and $f(D) > f(BD)$ .

**Definition 8.** An itemset $X$ in a database $D$ is a skyline frequent-utility pattern (SFUP) iff it is not dominated by any other itemset in the database when considering both the frequency and utility measures.

**Problem statement:** Based on the above definitions, we define the problem of skyline frequent-utility pattern mining (SFUPM) in a database as discovering the set of non-dominated itemsets by considering both the frequency and utility measures. Notice that the utility and frequency are treated as the same importance in this paper but can be adjusted by users' preference.

For instance, in the running example, the utilities and frequencies of the itemsets $(D)$, $(AD)$, and *(ABDE)* are respectively calculated as $u(D)$ $(= 25)$ and $f(D)$ $(= 5)$, $u(AD)$ $(= 28)$ and $f(AD)$ $(= 3)$, and *u(ABDE)* $(= 37)$ and *f(ABDE)* $(= 2)$. Those itemsets are the SFUPs for the example database since they are not dominated by any other itemsets in that database.

## 4. Proposed SFU-miner algorithm

In this section, an efficient algorithm called SFU-Miner is designed for mining the set of skyline frequent-utility patterns (SFUPs). It is based on the well-known utility-list structure. This structure allows quickly combining itemsets using a simple join operation. Details related to this structure are given next.

### *4.1. Utility-list structure*

Let $\triangleright$ be the ascending order of TWU values on items of a database. The utility-list [22] of an itemset $X$ in a database $D$ is a set of tuples, in which each tuple consists of three fields denoted as (*tid*, *iutil*, *rutil*). The $tid$ is the ID of a transaction containing the itemset $X$. The *iutil* and *rutil* elements of a tuple respectively are the utility of $X$ in the transaction $tid$; i.e., $u(X, T_{tid})$ and the remaining utility of the items except the itemset $X$ in $tid$, which can be defined as: $\sum_{i_j \in T_{tid} \wedge i_j \notin X} u(i_j, T_{tid})$.

For example, the utility-lists of items $(B)$ and $(D)$ are {($T_1$, 2, 5), ($T_2$, 2, 9), ($T_5$, 4, 10), ($T_6$, 8, 3)} and {($T_1$, 5, 0), ($T_2$, 5, 0), ($T_3$, 5, 0), ($T_5$, 5, 0), ($T_7$, 5, 0)}, respectively. The *twu*-ascending order of items for the running example is (*E*$\triangleright$*C*$\triangleright$*B*$\triangleright$*A*$\triangleright$*D*). For the example of Table 1, the constructed utility-lists are shown in Fig. 2.

Using the utility-lists of 1-itemsets (single items), the utility-lists of any $k$-itemset ($k \geqslant 2$) can be easily built by performing a simple join operation. According to the problem statement, an itemset is considered as a SFUP if it is not dominated by any other itemsets, when considering both the frequency and utility measures. To maintain information about the maximal utilities for different frequency (support) values, an utility-max (*umax*) array is initialized by the designed algorithm.

| ***E*** | | |
|---|---|---|
| *tid* | *iutil* | *rutil* |
| 2 | 4 | 14 |
| 4 | 4 | 2 |
| 5 | 8 | 14 |

| ***C*** | | |
|---|---|---|
| *tid* | *iutil* | *rutil* |
| 1 | 2 | 7 |
| 2 | 3 | 11 |
| 3 | 2 | 9 |
| 4 | 2 | 0 |
| 6 | 1 | 11 |

| ***B*** | | |
|---|---|---|
| *tid* | *iutil* | *rutil* |
| 1 | 2 | 5 |
| 2 | 2 | 9 |
| 5 | 4 | 10 |
| 6 | 8 | 3 |

| ***A*** | | |
|---|---|---|
| *tid* | *iutil* | *rutil* |
| 2 | 4 | 5 |
| 3 | 4 | 5 |
| 5 | 5 | 5 |
| 6 | 3 | 0 |

| ***D*** | | |
|---|---|---|
| *Tid* | *lutil* | *Rutil* |
| *1* | *5* | *0* |
| *2* | *5* | *0* |
| *3* | *5* | *0* |
| *5* | *5* | *0* |
| *7* | *5* | *0* |

Fig. 2. The constructed utility-lists for the running example.

**Definition 9.** An *umax* array stores the maximal utility for each frequency value $r$, which is denoted as *umax(r)*.

**Definition 10.** An itemset $X$ is considered as a potential SFUP (PSFUP) if its frequency is equal to $r$ and no itemsets have a utility greater than $u(X)$.

For example, the frequency of $(AD)$ is 3, and its utility is 28. Since there does not exist an itemset having a frequency of 3 and a utility greater than 28, the itemset $(AD)$ is considered as a PSFUP, and the maximal utility for the frequency $(= 3)$ is set to *umax(3)* $(= 28)$ .

### *4.2. Pruning strategy*

To mine SFUPs, the SKYMINE algorithm utilizes the UP-tree structure. A drawback of SKYMINE is that it often generates a large number of candidates. The reason is that it utilizes the two-phase model to overestimate the utilities of itemsets. To avoid this problem and speed up the discovery of SFUPs, we define two strategies.

**Lemma 1** (SFUPs $\subseteq$ PSFUPs)**.** If an itemset is not a PSFUP, it is also not a SFUP. Thus, all SFUPs are not PSFUPs.

*Proof.*

(1) $\forall X \notin$ PSFUPs, according to Definition 10, there exists an itemset $Y$ such that $f(Y) = f(X)$ and that its utility is greater than $u(X)$. Thus:
$\because u(Y) > u(X)$ and $f(Y) = f(X) \Rightarrow Y$ dominates $X$.
$\therefore X \notin$ *SFUPs*.

(2) $\forall X \in SFUPs$, according to Definitions 8 and 9, $X$ is not dominated by other itemsets.
$\forall\, Y$, $f(Y) = f(X)$, $X$ is not dominated by $Y$.
Thus, $u(X) > u(Y) \Rightarrow X \in$ *PSFUPs*.

□

For instance, the set of PSFUPs for the running example is *(ECBAD)*, *(EBAD)*, *(AD)*, $(A)$, $(B)$, and $(D)$, which have utilities and frequencies of *u(ECBAD)* $(= 18)$, *f(ECBAD)* $(= 1)$; *u(EBAD)* $(= 37)$, *f(EBAD)* $(= 2)$; *u(AD)* $(= 28)$, *f(AD)* $(= 3)$; *u(A)* $(= 16)$, *f(A)* $(= 4)$; *u(B)* $(= 16)$, *f(B)* $(= 4)$ and *u(D)* $(=$

25), *f(D)* (= 5), respectively. The set of SFUPs is *(EBAD)*, *(AD)* and $(D)$, which is a subset of the set of PSFUPs.

**Pruning strategy:** Let $X$ be an itemset. Moreover, let the extensions of $X$ be the itemsets obtained by appending an item $Y$ to $X$, where $X \rhd Y$. If the sum of *iutil* and *rutil* values in the utility-list of $X$ is less than *umax(r)* for $r = f(X)$, then all the extensions of $X$ are not SFUPs.

*Proof.* Let $X'$ be an extension of $X$, $\forall$ transaction $t \supseteq X' \Rightarrow (X' - X) = (X'/X)$
$\because X \subset X' \subseteq t \Rightarrow (X'/X) \subseteq (t/X)$.

$$\begin{aligned}\therefore f(X') \leqslant f(X), \text{ and } u(X', t) &= u(X, t) + u(X' - X, t)\\ &= u(X, t) + u(X'/X, t)\\ &= u(X, t) + \sum_{i_j \in X'/X} u(i_j, t)\\ &\leqslant u(X, t) + \sum_{i_j \in (t/X)} u(i_j, t)\\ &= u(X, t) + ru(X, t).\end{aligned}$$

Suppose that *tid(t)* denotes the *tid* of a transaction $t$, *t.tids* denotes the *tids* of tuples in the utility-list of $X$, and that *X′.tids* the *tids* of tuples in the utility-list of $X'$. Thus:
$\because X \subset X' \Rightarrow$ *X′.tids* $\subseteq$ *X.tids*.

$$\begin{aligned}\therefore u(X') = \sum_{id(t) \in X'.tids} u(X', t) &\leqslant \sum_{id(t) \in X'.tids} u(X,t) + rutil(X,t)\\ &\leqslant \sum_{id(t) \in X.tids} u(X,t) + rutil(X,t)\\ &< umax(f(X)).\end{aligned}$$

□

For each itemset $X$, it can be found that the frequencies of its extensions are always less than or equal to the frequency of $X$. Moreover, based on the above proof, it can be found that if the sum of *iutil* and *rutil* values in the utility-list of $X$ is less than *umax(r)* for $r = f(X)$, then there must be an itemset $Y$ having the same frequency as $X$ and the same utility *umax(r)*, i.e. Y dominates $X'$. This indicates that $X'$ is not a SFUP. For example, the itemset $(EA)$ has *umax(2)* = 37 and the utility-list of $(EA)$ is {($T_2$, 8, 5), ($T_5$, 13, 5)}. The sum of all the *iutil* and *rutil* values in its utility-list is 31, which is less than *umax(2)* (= 37). Therefore, there is no need to consider the extensions of the itemset *(EA)* (they are not SFUPs).

### *4.3. Proposed algorithm*

The designed SFU-Miner algorithm performs the following steps (listed in Algorithm 1). It first build the utility-list structure. The transaction-weighted utilities (*twu*) of all 1-itemsets are discovered by scanning the database. The 1-itemsets in the database are sorted by *twu*-ascending order. The sorted database is then used to construct the initial utility-lists of 1-itemsets.

In the designed SFU-Miner algorithm, the transaction-weighted utility of each item is first calculated and the database is then reorganized according to the ascending order of transaction-weighted utilities of 1-itemsets. The database is then scanned again to construct the utility-lists (*ULs*) of all 1-itemsets. The array *umax* is then used to prune unpromising itemsets during the mining process. Besides, the

**Algorithm 1:** SFU-Miner

**Input:** $D$, a transactional database; *ptable*, a profit table;
**Output:** The set of skyline frequent-utility patterns (SFUPs).

```
for i_j ⊆ T_q ∧ T_q ∈ D do
    calculate the tu of T_q;
    calculate twu(i_j) using T_q;
    calculate f(i_j);
for i_j ⊆ T_q ∧ T_q ∈ D do
    sort items in T_q in twu-ascending order;
    build the utility-list UL of i_j;
initialize an array with umax(r) := 0 for each r such that 1 ⩽ r ⩽ max{f(i_j)};
set PSFUPs := null;
set SFUPs := null ;
P-Miner(null, ULs, umax; PSFUPs);
SFUPs ←S-Miner (PSFUPs);
return SFUPs;
```

sets of *SFUPs* and *PSFUPs* are initalized as the empty set. They are used by the algorithm to store the current *SFUPs* and *PSFUPs* found by the algorithm, respectively. The procedure called **P-Miner** mines all potential SFUPs and stores them in the set of *PSFUPs*. After that, the itemsets in the set *PSFUPs* are evaluated to keep only the actual *SFUPs*. This is done by the procedure **S-Miner** shown in Algorithm 2.

**Algorithm 2:** P-Miner

**Input:** $P.UL$, the utility-list of the itemset $P$; *P'ULs*, the set of utility-lists of $P's$ extensions; *umax*, an array to keep the maximum utility for each frequency.
**Output:** *PSFUP*s; the set of $P's$ extensions that are potential skyline frequent-utility itemsets.

```
for each X in P'ULs do
    if sum(X.iutil) ⩾ umax(f(X)) then
        umax(f(X)) ← sum(X.iutil);
        PSFUIs ← X;
        remove Y from PSFUIs if (f(Y) == f(X));
    if sum(X.iutil) + sum(X.rutil) ⩾ umax(f(X)) then
        exULs := null;
        for each utility-list Y after X in P'ULs do
            exULs := exULs + construct(P.UL.X,Y);
        P-Miner(X, exULs, umax, PSFUPs);
```

Algorithm 2 shows the pseudo-code for mining potential SFUPs. For each utility-list $X$ in *ULs*, if the sum of all the *iutil* values in $X$ is greater than *umax(f(X))*, $X$ is a potential *SFUP*. If the sum of the *iutil* and *rutil* values of $X$ is greater than *umax(f(X))*, the extensions of $X$ should be further processed. The set of utility-lists of all the 1-extensions of itemset $X$ are recursively processed by intersecting the

| ***EC*** | | |
|---|---|---|
| *tid* | *iutil* | *rutil* |
| 2 | 7 | 11 |
| 4 | 6 | 0 |

| ***EB*** | | |
|---|---|---|
| *tid* | *iutil* | *rutil* |
| 2 | 6 | 9 |
| 5 | 12 | 10 |

| ***EA*** | | |
|---|---|---|
| *tid* | *iutil* | *rutil* |
| 2 | 8 | 5 |
| 5 | 13 | 5 |

| ***ED*** | | |
|---|---|---|
| *tid* | *iutil* | *rutil* |
| 2 | 9 | 0 |
| 5 | 13 | 0 |

Fig. 3. The utility-lists of $(E)$'s extensions.

utility-list $X$ with each utility-list $Y$ after $X$ in $ULs$. After the set of *PSFUPs* is discovered, the SFU-Miner algorithm is then executed to find the actual *SFUPs* in the set of *PSFUPs*. The result is presented to the user. It is the set of non-dominated itemsets (SFUPs). The pseudo-code of the **S-Miner** procedure is shown in Algorithm 3.

**Algorithm 3:** S-Miner

**Input:** *PSFUPs*, the set of potential SFUPs.
**Output:** *SFUPs*, the set of skyline frequent-utility patterns.

```
for each X ∈ PSFUPs do
    for each Y ∈ PSFUPs do
        if u(X) ⩾ u(Y) ∧ f(X) > f(Y) || u(X) > u(Y) ∧ f(X) ⩾ f(Y) then
            SFUPs ← X ∪ SFUPs ;
            remove Y from PSFUPs;
return SFUPs;
```

## 5. An illustrated example

This section provides a detailed example of how the proposed algorithm is applied on the database of Table 1 and the profit table shown in Table 2. The quantitative database of Table 1 is first scanned to find the transaction-weighted utilities of items. Those are shown in Table 3. After that, the items in Table 3 are sorted in $twu$-ascending order of transaction-weighted utilities. The result is shown in Table 4. In this example, the maximum frequency of the items $(A)$, $(B)$, $(C)$, $(D)$, $(E)$ is five. Thus, entries 1 to 5 in the *umax* array are initialized to 0.

The sorted database of Table 4 is then used to build the utility-lists of all 1-itemsets. The result is shown in Fig. 2. Consider the itemset $(E)$ as example to illustrate the following steps. The frequency of $(E)$ is calculated. It is found to be 3 since $(E)$ appears in transactions $T_2$, $T_4$, and $T_5$. The utility of $(E)$ is calculated as $u(E) = 4 + 4 + 8$ $(= 16)$. The *umax* value for the frequency 3 is then updated as *umax(3)* $= 16$. The other 1-itemsets are also processed in the same way. After that, the updated *umax* values are shown in Table 5.

The pruning strategy is then applied to determine if extensions of each 1-itemset should be considered by the depth-first search. In this example, since the sum of the *iutil* and *rutil* values of $(E)$ is 46, which is greater than *umax(3)* $(= 16)$, extensions of $(E)$ are generated. The utility-lists of the 1-extensions of $(E)$, which are *(EC)*, *(EB)*, *(EA)* and *(ED)*, are shown in Fig. 3.

It is found that the frequency of *(EC)* is 2 and that the utility of *(EC)* is 13, which is greater than the current value *umax(2)* $(= 0)$. Hence, the *umax(2)* value in Table 5 is updated to 13 and *(EC)* is then

Table 3
Transaction-weighted utilities of items

| Item | $A$ | $B$ | $C$ | $D$ | $E$ |
|---|---|---|---|---|---|
| *twu* | 63 | 61 | 56 | 65 | 46 |

Table 4
The database after sorting transactions by twu-ascending order

| TID | Items | TU |
|---|---|---|
| 1 | ($C$, 2) ($B$, 1) ($D$, 1) | 9 |
| 2 | ($E$, 1) ($C$, 3) ($B$, 1) ($A$, 4) ($D$, 1) | 18 |
| 3 | ($C$, 2) ($A$, 4) ($D$, 1) | 11 |
| 4 | ($E$, 1) ($C$, 2) | 6 |
| 5 | ($E$, 2) ($B$, 2) ($A$, 5) ($D$, 1) | 22 |
| 6 | ($C$, 1) ($B$, 4) ($A$, 3) | 12 |
| 7 | ($D$, 1) | 5 |

Table 5
The updated *umax* array

| Frequency | *umax* |
|---|---|
| 1 | 0 |
| 2 | 0 |
| 3 | 16 |
| 4 | 16 |
| 5 | 25 |

Table 6
The SFUPs

| Itemset | Frequency | *umax* |
|---|---|---|
| (*EBAD*) | 2 | 37 |
| (*AD*) | 3 | 28 |
| (*D*) | 5 | 25 |

considered to be a PSFUP. The other 2-itemsets are processed in the same way. After all extensions of $(E)$ have been checked, results are shown in Fig. 4.

Besides, the *umax* values are updated to *umax(1)* (= 18), *umax(2)* (= 37), *umax(3)* (= 16), *umax(4)* (= 16), and *umax(5)* (= 25), respectively. After all itemsets have been considered, the final set of SFUPs has been obtained, which is shown in Table 6.

## 6. Experimental results

This section presents the result of substantial experiments to evaluate the proposed SFU-Miner algorithm for mining SFUPs in several datasets. The performance is evaluated in terms of runtime, memory usage, number of candidates, and scalability. Notice that only one algorithm called SKYMINE [11] was previously proposed to mine SFUPs by considering both the frequency and the utility of itemsets. All the algorithms were implemented in Java and experiments were carried out on a personal computer equipped with an Intel Core2 i3-4160 CPU and 4GB of RAM, running the 64-bit Microsoft Windows 7 operating system. Four real-world datasets called chess [1], mushroom [1], foodmart [23] and retail [1], as well as five synthetic datasets, denoted as T10I4N4KDXK (where $X$ is varied from 100 to 500) [4], were used in the experiments to evaluate the performance of the proposed algorithm. In the experiments,

Table 7
Parameters to describe the datasets

| | |
|---|---|
| #\|**D**\| | Total number of transactions |
| #\|**I**\| | Number of distinct items |
| **AvgLen** | Average transaction length |
| **MaxLen** | Maximum transaction length |
| **Type** | Sparse or dense |

Table 8
Characteristics of the datasets

| Dataset | #\|**D**\| | #\|**I**\| | AvgLen | MaxLen | Type |
|---|---|---|---|---|---|
| Chess | 3,196 | 76 | 37 | 37 | Dense |
| Mushroom | 8,124 | 120 | 23 | 23 | Dense |
| Foodmart | 21,557 | 1,550 | 4 | 11 | Sparse |
| Retail | 88,162 | 16,470 | 10 | 76 | Sparse |
| T10I4N4KD100K | 98,307 | 3,999 | 10 | 28 | Sparse |

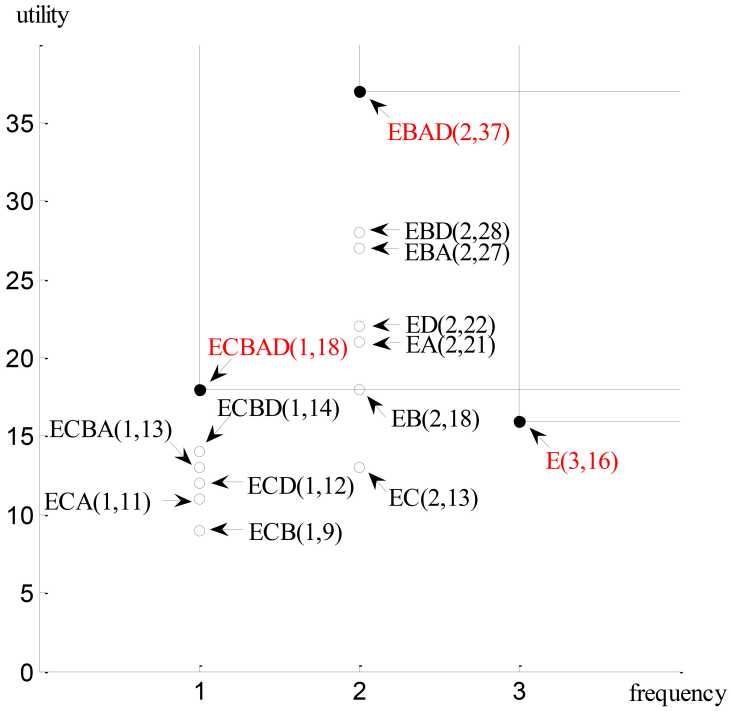


Fig. 4. The extensions of the itemset $(E)$.

algorithms were terminated if their runtime exceeded two hours or ran out of memory. Parameters and characteristics of these datasets are respectively shown in Tables 7 and 8.

### *6.1. Runtime*

This subsection compares the runtimes of the proposed algorithm and the SKYMINE algorithm [11] on five datasets. Results are given in Table 9.

In Table 9, it can be observed that there are no results for the SKYMINE algorithm on the chess and

Table 9
Runtimes of the compared algorithms

| | SFU-Miner | SKYMINE |
|---|---|---|
| Chess | 202.69 s | – |
| Mushroom | 10.57 s | 202.82 s |
| Foodmart | 2.64 s | 98.59 s |
| Retail | 117.57 s | – |
| T10I4N4KD100K | 57.32 s | 346.45 s |

Table 10
Memory usage of the compared algorithms

| | SFU-Miner | SKYMINE |
|---|---|---|
| Chess | 137.00 M | – |
| Mushroom | 18.87 M | 423.80 M |
| Foodmart | 32.13 M | 667.46 M |
| Retail | 143.68 M | – |
| T10I4N4KD100K | 234.30 M | 446.69 M |

retail datasets. The reason is that SKYMINE ran out of memory for those two datasets. We also observed that the proposed SFU-Miner algorithm outperforms the SKYMINE algorithm and is generally about one to two orders of magnitude faster than the SKYMINE algorithm. For example, for the mushroom dataset shown in Table 9, the runtimes of the SFU-Miner and SKYMINE algorithms are 10.57 and 202.82 seconds, respectively. This is reasonable since the proposed algorithm utilizes the utility-list structure and uses the *umax* array to keep the maximal utility of each frequency, to mine SFUPs efficiently. The SFU-Miner algorithm always outperforms the SKYMINE algorithm since the SFU-Miner algorithm can directly calculate the actual utility of itemsets by scanning the database only twice. The SKYMINE algorithm generates, however, many redundant candidates because it overestimates the utility of itemsets using the two-phase model. As a consequence, the SKYMINE algorithm requires more time for generating the candidates and determining the actual SFUPs.

### *6.2. Memory usage*

This subsection compares the memory usage of the proposed algorithm and the SKYMINE algorithm. Memory measurements were done using the Java API. The results for all datasets are given in Table 10.

In Table 10, no results are reported in terms of memory usage for the SKYMINE algorithm on the chess and retail datasets since it ran out of memory. It can be clearly seen that the proposed SFU-Miner algorithm requires less memory than the SKYMINE algorithm on all datasets. For example, the proposed SFU-Miner algorithm consumes 18.87 MB on the mushroom dataset while the SKYMINE algorithm utilizes 423.80 MB. This result is reasonable since the proposed SFU-Miner algorithm relies on the utility-list structure to directly mine SFUPs, and unpromising candidates can be easily pruned using the developed pruning strategy. The SKYMINE algorithm mines, however, the SFUPs based on the UP-Growth algorithm, which requires to generate numerous candidates. For this reason, it is not an efficient algorithm for mining the SFPUs.

### *6.3. Number of candidates*

In this subsection, the number of candidates produced by the proposed algorithm and the SKYMINE algorithm to discover SFUPs is compared on all datasets. The detailed results in terms of number of SFUPs and number of candidates are given in Table 11.

Table 11
Numbers of SFUs and PSFUs

| | SFUPs | PSFUs | |
|---|---|---|---|
| | | SFU-Miner | SKYMINE |
| Chess | 33 | 3,076 | – |
| Mushroom | 4 | 1,384 | 12,916 |
| Foodmart | 6 | 61 | 1,214,466 |
| Retail | 2 | 832 | – |
| T10I4N4KD100K | 4 | 879 | 7,994,015 |

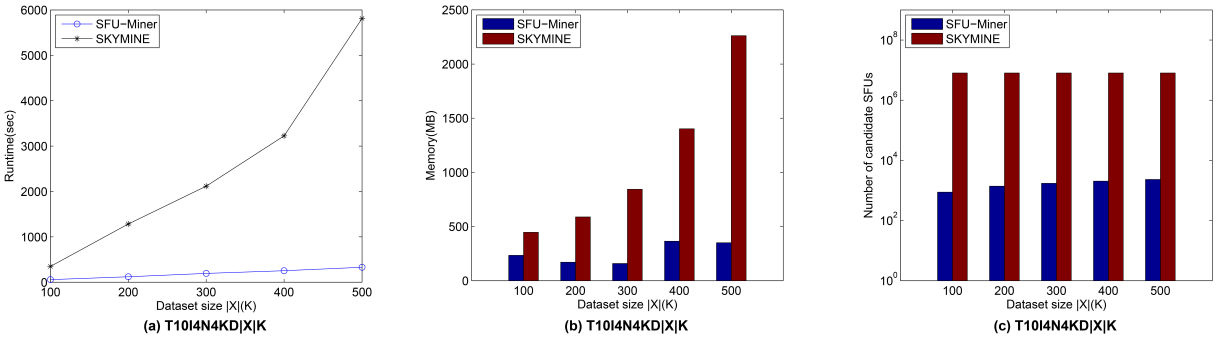


Fig. 5. Scalability of the compared algorithms.

It can be seen in Table 11 that the number of candidates produced by the SFU-Miner algorithm is generally about three to four orders of magnitude smaller than that of the SKYMINE algorithm. For example, the number of candidates produced by the designed SFU-Miner algorithm and the SKYMINE algorithms on the foodmart dataset are 61 and 1,214,466, respectively. This is reasonable since the SFU-Miner algorithm can prune unpromising candidates early using its developed pruning strategy, while the SKYMINE algorithm generates numerous candidates for mining SFUPs. For example, on the foodmart dataset, the designed algorithm finds 6 SFUPs from a set of 61 candidates, while SKYMINE finds 1,214,466 candidates to then derive 6 SFUP (as it can be seen in Table 11). Thus, it can also be observed that the SKYMINE algorithm requires more time and memory than the developed SFU-Miner algorithm since SKYMINE produces more candidates for mining the actual SFUPs.

### *6.4. Scalability*

The scalability of the proposed algorithm and the SKYMINE algorithm was compared on the synthetic dataset T10I4N4KD|X|K by varying the dataset size from 100 K to 500 K transactions. The runtime, memory usage, and number of candidates were measured. Results are shown in Fig. 5.

In Fig. 5, it can be observed that the designed SFU-Miner algorithm has better scalability than the SKYMINE algorithm in terms of runtime, memory usage, and number of candidates, when the dataset size is varied. The proposed SFU-Miner algorithm always outperforms the SKYMINE algorithm in terms of runtime and memory usage. For example, the developed SFU-Miner algorithm is about one to two orders of magnitude faster than the SKYMINE algorithm when the dataset size is increased from 100 K to 500 K transactions, as shown in Fig. 5a. An interesting observation is that the runtime and memory consumption of the proposed SFU-Miner algorithm steadily increases as the number of transactions increases, while the runtime and memory usage of SKYMINE sharply increases as the dataset size increases. Moreover, the SKYMINE algorithm produces three to four orders of magnitude

more candidates for mining the SFUPs compared to the designed SFU-Miner algorithm, as shown in Fig. 5c. From the results of this scalability experiment, it can be concluded that the proposed algorithm has good scalability in terms of runtime, memory usage and number of candidates, for mining SFUPs.

## 7. Conclusion and future work

In the past, many approaches have been proposed to mine frequent itemsets or high utility itemsets in transaction databases. However, to the best of our knowledge, few studies have jointly considered the measures of frequency and utility to select a set of patterns that is both useful and frequent. This paper has proposed a novel algorithm called SFU-Miner to mine the set of skyline frequent-utility itemsets by considering both the frequency and utility measures. The designed SFU-Miner algorithm relies on the utility-list structure and the *umax* array for mining SFUPs. A pruning strategy has further been designed to prune unpromising candidates early for deriving the SFUPs efficiently. Using the designed algorithm, a user does not need to set a minimum support threshold or a minimum utility threshold to discover the set of all dominant patterns in the utility and frequency dimensions. Substantial experiments were conducted on both real-life and synthetic datasets to assess the performance of the proposed algorithm in terms of runtime, memory usage, number of candidates, and scalability. Results have shown that the proposed algorithm largely outperforms SKYMINE, the previous state-of-the-art algorithm for SFUP mining.

Since few studies have considered the issue of skyline frequent-utility pattern mining, there are numerous opportunities to extend the techniques presented in this paper to other problems, such as stream mining, top-*k* pattern mining or dynamic data mining. Besides, more efficient and condensed structures for mining SFUPs in various real-life applications can be further discussed and studied.

## Acknowledgments

This research was partially supported by the National Natural Science Foundation of China (NSFC) under grant No. 61503092, by the Shenzhen Technical Project under JCYJ20170307151733005, by the Science Research Project of Guangdong Province under grant No. 2017A020220011, and by the National Science Funding of Guangdong Province under Grant No. 2016A030313659.

## References

[1] Frequent itemset mining dataset repository http://fimi.ua.ac.be/data/ (2012).
[2] F. Afrati, P. Koutris, D. Suciu and J.D. Ullman, Parallel skyline queries, *Theory of Computing Systems* **57**(4) (2015), 1008–1037.
[3] R. Agrawal and R. Srikant, Fast algorithm for mining association rules, *International Conference on Very Large Data Bases* (1994), pp. 487–499.
[4] R. Agrawal and R. Srikant, Quest synthetic data generator, http://www.Almaden.ibm.com/cs/quest/syndata.html, 1994.
[5] C.F. Ahmed, S.K. Tanbeer, B.S. Jeong and Y.K. Le, Efficient tree structures for high utility pattern mining in incremental databases, *IEEE Transactions on Knowledge and Data Engineering* **21**(12) (2009), 1708–1721.
[6] S. Borzsonyi, D. Kossmann and K. Stocker, The skyline operator, *International Conference on Data Engineering* (2001), pp. 421–430.
[7] R. Chan, Q. Yang and Y.D. Shen, Mining high utility itemsets, *IEEE International Conference on Data Mining* (2003), pp. 19–26.
[8] C.Y. Chan, H.V. Jagadish, K.L. Tan, A.K.H. Tung and Z. Zhang, Finding k-dominant skylines in high dimensional space, *ACM SIGMOD International Conference on Management of Data* (2006), pp. 503–514.

[9] J. Chomicki, P. Godfrey, J. Gryz and D. Liang, Skyline with presorting, *International Conference on Data Engineering* (2003), pp. 717–720.
[10] P. Fournier-Viger, C.W. Wu, S. Zida and V.S. Tseng, FHM: Faster high-utility itemset mining using estimated utility co-occurrence pruning, *Foundations of Intelligent Systems* **8502** (2014), 83–92.
[11] V. Goyal, A. Sureka and D. Patel, Efficient skyline itemsets mining, *The International C* Conference on Computer Science & Software Engineering* (2015), pp. 119–124.
[12] G. Grahne and J. Zhu, Efficiently using prefix-trees in mining frequent itemsets, *IEEE ICDM Workshop on Frequent Itemset Mining Implementations*, 2003.
[13] J. Han, J. Pei and Y. Yin, Mining frequent patterns without candidate generation, *ACM SIGKDD International Conference on Management of Data* (2000), pp. 1–12.
[14] D. Kossmann, F. Ramsak and S. Rost, Shooting stars in the sky: an online algorithm for skyline queries, *International Conference on Very Large Data Bases* (2002), pp. 275–286.
[15] X. Lin, Y. Yuan, W. Wang and H. Lu, Stabbing the sky: efficient skyline computation over sliding windows, *International Conference on Data Engineering* (2005), pp. 502–513..
[16] X. Lin, Y. Yuan, Q. Zhang and Y. Zhang, Selecting stars: the $k$ most representative skyline operator, *International Conference on Data Engineering* (2007), pp. 86–95.
[17] C.W. Lin, T.P. Hong and W.H. Lu, The Pre-FUFP algorithm for incremental mining, *Expert Systems with Applications* **36**(5) (2009). 9498–9505.
[18] C.W. Lin, T.P. Hong and W.H. Lu, An effective tree structure for mining high utility itemsets, *Expert Systems with Applications* **38**(6) (2011), 7419–7424.
[19] C.W. Lin, T.P. Hong, G.C. Lan, J.W. Wong and W.Y. Lin, Efficient updating of discovered high-utility itemsets for transaction deletion in dynamic databases, *Advanced Engineering Informatics* **29**(1) (2015), 16–27.
[20] J. Liu, Y. Pan and K. Wang, Mining frequent itemsets by opportunistic projection, *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (2002), pp. 229–238.
[21] Y. Liu, W.K. Liao and A. Choudhary, A two-phase algorithm for fast discovery of high utility itemsets, *Lecture Notes in Computer Science* **3518** (2005), 689–695.
[22] M. Liu and J. Qu, Mining high utility itemsets without candidate generation, *ACM International Conference on Information and Knowledge Management* (2012), pp. 55–64.
[23] Microsoft, Example database foodmart of Microsoft analysis services, http://msdn.microsoft.com/en-us/library/aa217032(SQL.80).aspx.
[24] D. Papadias, Y. Tao and B. Seeger, Progressive skyline computation in database systems, *ACM Transaction on Database Systems* **30**(1) (2005), 41–82.
[25] J.S. Park, M.S. Chen and P.S. Yu, An effective hash based algorithm for mining association rules, *ACM SIGMOD International Conference on Management of Data* (1995), pp. 175–186.
[26] J. Pei, J. Han and H. Lu, H-mine: hyper-structure mining of frequent patterns in large databases, *IEEE International Conference on Data Mining* (2001), pp. 441–448.
[27] V. Podpecan, N. Lavrac and I. Kononenko, A fast algorithm for mining utility-frequent itemsets, *International workshop on Constraint-Based Mining and Learning* (2007), pp. 9–20.
[28] A. Savasere, E. Omiecinskiand and S. Navathe, An efficient algorithm for mining association rules in large databases, *International Conference on Very Large Databases* (1995), pp. 432–444.
[29] K.L. Tan, P.K. Eng and B.C. Ooi, Efficient progressive skyline computation, *International Conference on Very Large Data Bases* (2001), pp. 301–310.
[30] V.S. Tseng, B.E. Shie, C.W. Wu and P.S. Yu, Efficient algorithms for mining high utility itemsets from transactional databases, *IEEE Transactions on Knowledge and Data Engineering* **25**(8) (2012), 1772–1786.
[31] H. Yao, H.J. Hamilton and C.J. Butz, A foundational approach to mining itemset utilities from databases, *SIAM International Conference on Data Mining* (2004), pp. 211–225.
[32] H. Yao, H.J. Hamilton and L. Geng, A unified framework for utility-based measures for mining itemsets, *ACM SIGKDD International Conference on Utility-Based Data Mining* (2006), pp. 28–37.
[33] H. Yao and H.J. Hamilton, Mining itemset utilities from transaction databases, *Data & Knowledge Engineering* **59**(3) (2006), 603–626.
[34] J.S. Yeh, Y.C. Li and C.C. Chang, Two-phase algorithms for a novel utility-frequent mining model, *International Conference on Emerging Technologies in Knowledge Discovery and Data Mining* (2007), pp. 433–444.
[35] S.J. Yen and Y.S. Lee, Mining high utility quantitative association rules, *Lecture Notes in Computer Science* **4654** (2007), 283–292.
[36] M. Zihayat and A. An, Mining top-$k$ high utility patterns over data streams, *Information Sciences* **285**(99) (2014), 138–161.