Engineering Applications of Artificial Intelligence 77 (2019) 229–238

Contents lists available at ScienceDirect

Engineering Applications of Artificial Intelligence

journal homepage: www.elsevier.com/locate/engappai

# Mining of skyline patterns by considering both frequent and utility constraints

Jerry Chun-Wei Lin [a,*], Lu Yang [b], Philippe Fournier-Viger [c], Tzung-Pei Hong [d,e]

[a] Department of Computing, Mathematics, and Physics, Western Norway University of Applied Sciences, Bergen, Norway
[b] School of Computer Science and Technology, Harbin Institute of Technology (Shenzhen), Shenzhen, China
[c] School of Natural Sciences and Humanities, Harbin Institute of Technology (Shenzhen), Shenzhen, 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

ARTICLE INFO

*Keywords:*
Data mining
Utility-list structure
Skyline frequent-utility patterns (SFUPs)
Maximal

ABSTRACT

Association-rule mining (ARM) or frequent itemset mining (FIM) is the most fundamental task in knowledge discovery, which is used to find the occurrence frequency of the item/sets in transactional database. The other factors such as weight, interestingness or unit profit of the items are not considered whether in ARM or FIM. To reveal more information, high-utility itemset mining (HUIM) was designed to consider both quantity and unit profit of items to discover the high-utility itemsets (HUIs). Several algorithms for FIM or HUIM were extensively studied but fewer works concern both frequency and utility together to provide better solutions in decision-making. In the past, the SKYMINE algorithm was designed to find the skyline frequent-utility patterns (SFUPs). A SFUP is a non-dominated pattern, in which each solution dominates the others by considering the aspects of frequency and utility. The SKYMINE algorithm needs, however, amounts of computation to level-wisely discover the SFUPs. In this paper, an efficient utility-list structure is used instead of the UP-tree structure used in SKYMINE to mine the SFUPs. Two algorithms are respectively designed by using the depth-first search (called SKYFUP-D) and breath-first search (SKYFUP-B) to mine the SFUPs. An efficient structure is also designed to record the maximal utility of the potential itemsets, thus reducing the computations for finding the SFUPs in the search space. Extensive experiments are conducted on several real-world and simulated datasets and the results indicate that the designed two algorithms have better performance than that of the state-of-the-art SKYMINE algorithm in terms of runtime, memory usage, search space size and the scalability.

## 1. Introduction

In traditional data mining techniques, association-rule mining (ARM) or frequent itemset mining (FIM) is the most common way (Agrawal et al., 1993; Agrawal and Srikant, 1994a; Park et al., 1995; Fournier-Viger et al., 2017; Zaki et al., 1997) to reveal the occurrence frequency of the itemsets. For the ARM, it first finds the set of frequent itemsets (FIs) against the minimum support threshold in the first step. The satisfied FIs are combined together to form the set of association rules against the minimum confidence threshold. This process is the level-wise approach, thus amounts of computations to generate the candidates in each level are necessary. For the ARM, it only reflects whether an item is bought in a transaction. The other factors such as weight, unit profit of the items, or interestingness are not considered in the ARM or FIM, thus the provided information for making the efficient decision to retailers is somehow insufficient. Besides, only the occurrence frequency cannot identify whether the discovered information is an important rule for retailers. For example, a sale of diamonds may occur less frequently than that of clothing in a department store, but the former item brings a much higher profit per unit sold than the latter one. Thus, the frequency of an item/set is not enough to identify highly profitable patterns.

High-utility itemset mining (HUIM) (Ahmed et al., 2009; Gan et al., 2018a,b; Yao et al., 2004; Yao and Hamilton, 2006; Yen and Lee, 2007) was proposed to solve the limitations of ARM or FIM. An item/set is considered as a high-utility itemset (HUI) if its utility is no less than the pre-defined minimum utility count. Chan et al. (2003) first proposed utility mining problem instead of FIM. Yao et al. (2004) concerned the quantity of items as the internal utility and the unit profit of items as the external utility to discover the set of HUIs. Since traditional HUIM does not consider the downward closure (DC) property, thus the process to mine the HUIs is a non-trivial task. Liu et al. (2005) then proposed a transaction-weighted utility (TWU) model to maintain the transaction-weighted downward closure (TWDC) property of the designed high transaction-weighted utilization itemsets (HTWUIs), thus reducing the search space for mining the HUIs. Lin et al. (2011) then designed the

* Corresponding author.
*E-mail addresses:* jerrylin@ieee.org (J.C.-W. Lin), luyang@ikelab.net (L. Yang), philfv@hitsz.edu.cn (P. Fournier-Viger), tphong@nuk.edu.tw (T.-P. Hong).

https://doi.org/10.1016/j.engappai.2018.10.010
Received 5 December 2016; Received in revised form 12 September 2018; Accepted 11 October 2018
Available online 30 October 2018

high-utility pattern (HUP)-tree to keep the necessary information in the tree-based structure for mining the HUIs. Several algorithms were extensively studied (Fournier-Viger et al., 2014; Liu and Qu, 2012; Tseng et al., 2010; Wu et al., 2012), which showed that the HUIM is an emerging topic in recent decades.

The above algorithms are required to generate a huge amount of rules for decision-making depending on how parameters are set. However, users are interested in discovering more concise rules since it takes time to select the most appropriate rules for making the efficient strategies. Top-*k* association-rule mining (Fournier-Viger et al., 2012) and high-utility mining (Tseng et al., 2015) are presented to mine the top-*k* rules from the databases. Thus, the users can only set the *k* most association rules or high-utility rules for decision-making. Although the rules for decision-making can be greatly reduced, only one aspect such as high confidence or high utility of rules is obtained to discover the required information. In real-life situations, it is necessary to consider two or more factors together for decision-making. For example, considering the distance to the destination and the price of the hotel, if the hotel is in the downtown, the price of the hotel in the downtown should be higher than that in the countryside. Mining whether top-*k* association rules or high-utility itemsets can only show the most *k* rules regarding to the confidence or utility aspect, which is insufficient to provide more flexible solutions by two or more aspects. Thus, it is better to provide a set of hotels (solutions) for the end-user to make the efficient decisions. In the past, Goyal et al. (2015) first proposed skyline frequent-utility (SFU) itemset mining algorithm called SKYMINE to discover the set of the non-dominated itemsets, in which each skyline point dominates the others under the aspects of frequency and utility. This process is based on the UP-tree structure (Tseng et al., 2010), thus amounts of candidates are required to be generated. Pan et al. (2017) presented the SFU-Miner to improve the performance for obtaining the skyline solutions but still face the limitation in terms of runtime. To solve the above limitations, efficient list-based skyline mining algorithms (named SKYFUP-D and SKYFUP-B) are presented in this paper. Major contributions are listed below.

1. Two efficient algorithms called SKYFUP-B (by breath-first search) and SKYFUP-D (by depth-first search) are presented to consider the patterns with higher frequency and utility, and return the set of non-dominated points as the solutions for decision-making.
2. The utility-max (*utilmax*) structure is designed to keep the maximal utility of the itemsets under the frequency constraint, thus reducing the search space to mine the required SFUPs.
3. Based on the designed algorithms, it is unnecessary to set up the minimum support threshold or minimum utility threshold for revealing the set of desired SFUPs but the set of solutions can be completely obtained under both occurrence frequency and utility constraints.
4. Extensive experiments on various databases were respectively performed to compare the proposed algorithm with the state-of-the-art SKYMINE algorithm for mining SFUPs and the results showed that the designed algorithms have better performance in terms of runtime, memory usage, search space size and the scalability.

The rest of this paper is organized as follows. Related work is discussed in Section 2. Preliminary and the problem statement of skyline frequent-utility pattern mining (SFUPM) are stated in Section 3. The proposed SKYFUP-D, SKYFUP-B algorithms and the designed *utilmax* structure to reduce the search space for mining SFUPs are presented in Section 4. An example to illustrate the proposed algorithms is given in Section 5. An experimental evaluation of the designed algorithms with the state-of-the-art SKYMINE approach is provided in Section 6. Conclusion and future work are finally mentioned in Section 7.

## 2. Related work

In this section, works related to high-utility itemset mining (HUIM) and skyline concept are briefly reviewed.

### 2.1. High-utility itemset mining

Frequent itemset mining (FIM) plays an important role in association rule mining (ARM) since the set of frequent itemsets (FIs) should be revealed as the first step in ARM. Extensive studies have been proposed for mining FIs and can be divided into two categories as: level-wise (Agrawal and Srikant, 1994a; Park et al., 1995; Savasere et al., 1995) and pattern-growth (Han et al., 2000; Liu et al., 2002; Pei et al., 2001) approaches. For the level-wise approaches, the candidates itemsets are tediously generated at each level, and the mostly common algorithm is called the Apriori algorithm (Agrawal and Srikant, 1994a). To improve the mining performance, the fundamental pattern-growth approach named frequent-pattern (FP)-growth (Han et al., 2000) was designed to mine FIs from the developed FP-tree structure. However, the above approaches can only reveal the occurrence frequency of the item/sets, but the other factors such as unit profit, weight or interestingness of the item/sets are not considered in ARM or FIM.

High-utility itemset mining (HUIM) is an extension of FIM, which considers both the quantity and unit profit of the item/sets in the database to mine the high-utility itemsets (HUIs). An itemset is defined as the HUI if its utility is no less than the pre-defined minimum utility threshold. Yao et al. (2006) proposed an algorithm for efficiently mining high utility itemsets based on the mathematical property of utility constraints. The designed algorithm cannot hold the downward closure (DC) property, thus the complete set of HUIs could not be discovered. Liu et al. (2005) then proposed a transaction-weighed utility (TWU) model to find the HUIs by holding the transaction-weighted downward closure (TWDC) property of the designed high transaction-weighted-utilization itemsets (HTWUIs). The TWU model requires, however, amounts of search space for mining the HUIs, and multiple database scans. Moreover, a novel list-based algorithm called HUI-Miner (Liu and Qu, 2012) was presented to mine the HUIs. It uses the simple join operation to find the set of the potential HUIs, and the number of candidates and the execution time for mining the HUIs can be greatly reduced. Each tuple of the utility-list structure keeps three elements, which respectively are the transaction id (*tid*) containing the itemset $X$, the utility value of an itemset $X$ in a transaction (*iutil*) and the remaining utility value of the itemset $X$ in a transaction (*rutil*). Several algorithms (Ahmed et al., 2009; Tseng et al., 2010, 2012) of HUIM were extensively proposed to efficiently reduce the search space for mining the HUIs.

Although the algorithms developed for FIM or HUIM can efficiently and effectively mine the desired information, fewer algorithms consider both the frequency and the utility as the aspects together to mine the required information. Yeh et al. (2007) first proposed the two-phase algorithm to discover the set of itemsets, in which the utility and the frequency of each itemset is no less than a user specified minimum utility threshold or minimum support threshold. Podpecan et al. (2007) proposed a fast algorithm to mine the set of utility-frequent itemsets but still has to define two thresholds. Since it is not a trivial task to set up the precisely thresholds for mining the required information, Goyal et al. proposed an efficient skyline itemset mining (SKYMINE) algorithm (Goyal et al., 2015) to discover the set of itemsets in which each itemset is non-dominated by the others. This approach is unnecessary to set up two thresholds but a set of skyline points are returned as the solutions for later decision-making. The SKYMINE algorithm is based on the UP-tree structure (Tseng et al., 2010), and amounts of candidates are required to be generated, which is not efficient for handling the large scale databases.

### 2.2. Skyline concept

To obtain better and concise rules from a huge amount of the discovered information, top-*k* association-rule mining (Fournier-Viger et al., 2012) or top-*k* high-utility itemsets mining (Tseng et al., 2015)

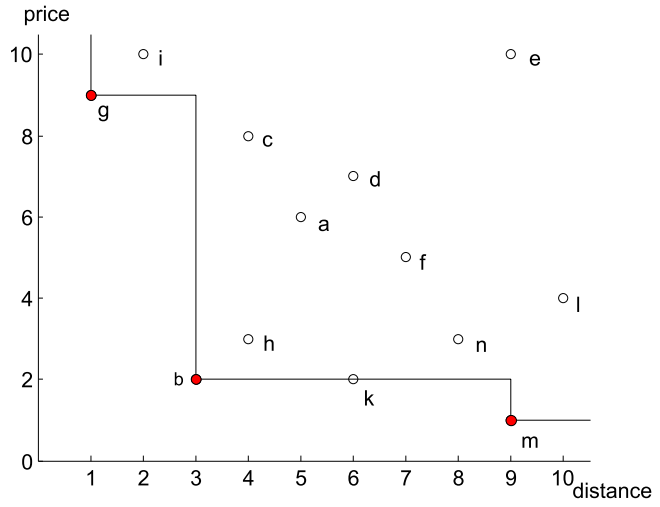



**Fig. 1.** Example dataset and its skylines.

were respectively presented. Those approaches can find the most $k$ information for decision-making regarding to only one aspect of minimum support or minimum high-utility thresholds. In practical example, users may need to consider more than one aspect to make the decision. For instance, a visitor or traveler may consider two factors such as the distance of the hotel to the downtown and price to book the hotel per night. The hotel is close to the downtown normally with higher price than the hotel in the countryside. Thus, it is not flexible to only consider one factor for decision-making. The traditional top-$k$ information is thus insufficient to obtain better solutions.

The skyline represents a set of points, and each point is non-dominated by the others based on multi-dimensions. It plays an important role for handling the large scale databases since only the non-dominated points are returned as the solutions for decision-making. For example, assume as set of $m$ objects, the skyline of $m$ objects can be referred to those are not dominated by any other object. If an object $X$ dominates another object $Y$, it indicates that $X$ is as good or better in all dimensions and better in at least on dimension than $Y$. In real-world cases, the distance from the hotel to the downtown and the price of the hotel is contrast; if a hotel is close to the downtown, the price for booking a room is higher than that of the hotel is far away from the downtown. An example can be illustrated in Fig. 1.

In Fig. 1, we have a set of hotels for booking. We keep the distance from hotel to the beach in $x$ axis, and the price of each hotel is marked in $y$ axis. In this example, we can observe that the skyline points are respectively $b$, $g$ and $m$, since they are the non-dominated points in Fig. 1 by considering two dimensions together.

The skyline concept was first proposed by Kung et al. (2005) using the divide-and-conquer approach to find the set of points. Borzsonyi et al. (2001) introduced the skyline operation for database context and used the B-tree and R-tree to evaluate the skyline points. Chomicki et al. (2003) then presented an improved version of the block nested loops to increase the performance by employing the certain ordering of tuples. Tan et al. (2001) proposed the algorithm to progressively output skyline points without scanning the entire input data. Kossmann et al. (2002) presented an NN algorithm based on the nearest-neighbor search, and applied the divide-and-conquer and R-tree techniques to find the skyline points. Papadias et al. (2005) proposed a branch-and-bound skyline (BBS) algorithm to perform a single access for finding the skyline points based on the nearest-neighbor search. Other related works were extensively studied (Afrati et al., 2015; Chan et al., 2006; Lin et al., 2007; Pei et al., 2005).

In the traditional HUIM or FIM, both of them consider one aspect for mining the desired information. The skyline method can, however, find the non-dominated solutions based on multi-dimensions. In this paper, list-based SKYFUP-D algorithm and SKYFUP-B algorithm are presented to consider the utility and the frequency factors together. The utility-list structure is used in this paper for easily performing the join operation to find the desired information. Besides, the utility-max (*utilmax*) structure is presented to further keep the maximal utility of several frequent values, thus reducing the search space for finding the skyline frequent-utility patterns (SFUPs). Preliminary and problem statement of the skyline frequent-utility pattern mining (SFUPM) are stated below.

**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, $D$:1, $E$:1 |
| 5 | $A$:5, $B$:2, $D$:1, $E$:2 |
| 6 | $A$:3, $B$:4, $C$:1, $D$:1 |
| 7 | $D$:1, $E$:1 |

**Table 2**
A profit table.

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

## 3. Preliminary 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 $T_q \in D$ $(1 \leq q \leq n)$ is a subset of $I$ and has a unique identifier $q$, called its $TID$. Besides, each item $i_j$ in a transaction $T_q$ has its purchase quantity (internal utility) and 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 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 $X \subseteq T_q$. An illustrated example stated in Table 1 is used as the running example in this paper. In Table 1, it has 7 transactions and 5 distinct items, respectively denoted from $(A)$ to $(E)$. The profit value (external utility) of each item is shown in Table 2 as the profit table.

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

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

For example in Table 1, the frequency of $(D)$ and $(BCD)$ are respectively calculated as 7 and 3 since $(D)$ appears in transactions $T_1$, $T_2$, $T_3$, $T_4$, $T_5$, $T_6$, and $T_7$ and $(BCD)$ appears in transactions $T_1$, $T_2$ and $T_6$.

**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:

$$u(i_j, T_q) = q(i_j, T_q) \times pr(i_j). \quad (2)$$

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

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

For example in Table 1, the utilities of the itemsets $(D)$ and $(BCD)$ in transaction $T_1$ are respectively calculated as $u(D,T_1) = q(D,T_1) \times pr(D) = 1 \times 5\ (= 5)$ and $u(BCD,T_1) = q(B,T_1) \times pr(B) + q(C,T_1) \times pr(C) + q(D,T_1) \times pr(D) = 1\ x\ 2 + 2\ x\ 1 + 1\ x\ 5 (= 9)$.

**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 in Table 1, the utility of the itemsets $(D)$ and $(BCD)$ in $D$ are respectively calculated as $u(D) = u(D,T_1) + u(D,T_2) + u(D,T_3) + u(D,T_4) + u(D,T_5) + u(D,T_6) + u(D,T_7) = 5+5+5+5+5+5+5 (= 35)$, and $u(BCD) = u(BCD,T_1) + u(BCD,T_2) + u(BCD,T_6) = 9 + 10 + 14 (= 33)$.

**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 in Table 1, $tu(T_1) = u(B,T_1) + u(C,T_1) + u(D,T_1) = 2 + 2 + 5\ (= 9)$. The resting transactions from $T_2$ to $T_7$ are calculated in the same way and the results are $tu(T_2)\ (= 18)$, $tu(T_3)\ (= 11)$, $tu(T_4)\ (= 11)$, $tu(T_5)\ (= 22)$, $tu(T_6)\ (= 17)$, and $tu(T_7)\ (= 9)$.

To maintain the downward closure (DC) property in HUIM, the transaction-weighted utility (*twu*) model was designed to maintain the transaction-weighted-utilization downward closure (TWUDC) property by using the *tu* value as the upper-bound of the itemset in the transaction. The definition is given below.

**Definition 6.** The transaction-weighted utility of an itemset $X$ in $D$ 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 in Table 1, $twu(D) = tu(D,T_1) + tu(D,T_2) + tu(D,T_3) + tu(D,T_4) + tu(D,T_5) + tu(D,T_6) + tu(D,T_7) = 9+18+11+11+22+17+9 (= 97)$.

To consider both the frequency and utility factors together, the definitions of the skyline frequent-utility pattern mining (SFUPM) are respectively given below.

**Definition 7.** An itemset $X$ dominates another itemset $Y$ in $D$, iff $f(X) \geq f(Y)$ and $u(X) > u(Y)$ or $f(X) > f(Y)$ and $u(X) \geq u(Y)$, which is denoted as $X \succ Y$.

For example in Table 1, the itemset $(D) \succ (BCD)$ since $u(D) > u(BCD)$ and $f(D) > f(BCD)$.

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

**Problem Statement:** Based on the above definitions, we define the problem of skyline frequent-utility pattern mining (SFUPM) as discovering the set of non-dominated itemsets in the database by considering both the frequency and utility factors.

For example in Table 1, the frequency and utility of $(D)$ are respectively calculated as 7 and 35; frequency and utility of $(DE)$ are respectively calculated as 4 and 40; and frequency and utility of $(ABD)$ are respectively calculated as 3 and 41. The $(D)$, $(DE)$, and $(ABD)$ can be considered as the SFUPs since any of them is non-dominated by other itemsets in the database.

## 4. Proposed SFUP mining algorithms

In the designed algorithms, the search space for mining the SFUPs can be explored by either depth-first search or breadth-first search. The reason is that the itemsets in the utility-list are sorted in *twu-ascending* order, thus the complete SFUPs can be explored. For an itemset $X$, the depth-first search (named SKYFUP-D) is to find the extensions (supersets) of $X$ in different level (or length) but the breath-first search (named SKYFUP-B) explores the itemsets with the same length of $X$. Besides, the utility-list (UL) structure is thus used in the designed algorithms to faster find the combined patterns with simple join operation. Details of the utility-list (UL) structure and the proposed algorithms are given below.

### 4.1. Utility-list (UL) structure

Let $\rhd$ be *twu-ascending* order of the items in the database $D$. The utility-list (UL) structure (Liu and Qu, 2012) of an itemset $X$ keeps the set of tuples and each tuple contains three elements as (*tid*, *iutil*, *rutil*), where the *tid* is the transaction ID containing the itemset $X$, the *iutil* is the actual utility of the itemset $X$ in transaction *tid* such as $u(X,T_q)$, and *rutil* is the summation of the utilities of all items after $X$ in *tid* such as $\sum_{i_j \in T_q/X} u(i_j, T_q)$.

For example in Table 1, the *twu-ascending* order of all items are $E \rhd B \rhd C \rhd A \rhd D$. The UL structure of itemset $(D)$ and $(BCD)$ respectively are $UL.D = \{(T_1, 5, 0), (T_2, 5, 0), (T_3, 5, 0), (T_4, 5, 0), (T_5, 5, 0), (T_6, 5, 0), (T_7, 5, 0)\}$ and $UL.BCD = \{(T_1, 9, 0), (T_2, 10, 0), (T_6, 14, 0)\}$. From the given example in Table 1, the constructed UL structures for all 1-itemsets are shown in Fig. 2.

### 4.2. The pruning strategy

**Definition 9.** The utility-max (*utilmax*) structure keeps the maximal utility among the patterns if their frequency is more than or equal to the index parameter $i$ $(1 \leq i \leq |D|)$ as:

$$utilmax(i) = max\{u(X) | f(X) \geq i\}. \tag{7}$$

Based on the *utilmax* structure, it can be used to keep the maximal utility of the itemsets under the frequency constraint and reduce the search space in the process for mining SFUPs.

**Lemma 1.** *Let $X$ be an itemset. If the summed up of iutil in the UL structure of $X$ is less than the $utilmax(f(X))$, $X$ must not be a SFUP.*

**Proof.** For an itemset $Y$ such that $f(Y) \geq f(X)$ and $u(Y) = utilmax(f(X))$. $\because \sum_{X \subseteq T_q \wedge T_q \in D} util(X, T_q) < utilmax(f(X))$, $\therefore u(X) < u(Y)$.

Since $f(Y) \geq f(X)$, thus $Y$ dominates $X$, which means $X$ must not be a SFUP. □

**Lemma 2.** *Let $X$ be an itemset, and let the extensions of $X$ by appending an item $Y$ to $X$ as $(X \cup Y)$ such that $X \rhd Y$. If the sum of iutil and rutil values in the utility-list of $X$ is less than $utilmax(f(X))$, then all the extensions of $X$ are not SFUPs.*

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

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

| (E) | | |
|---|---|---|
| *tid* | *iutil* | *rutil* |
| 2 | 4 | 14 |
| 4 | 4 | 7 |
| 5 | 8 | 14 |
| 7 | 4 | 5 |

| (B) | | |
|---|---|---|
| *tid* | *iutil* | *rutil* |
| 1 | 2 | 7 |
| 2 | 2 | 12 |
| 5 | 4 | 10 |
| 6 | 8 | 9 |

| (C) | | |
|---|---|---|
| *tid* | *iutil* | *rutil* |
| 1 | 2 | 5 |
| 2 | 3 | 9 |
| 3 | 2 | 9 |
| 4 | 2 | 5 |
| 6 | 1 | 8 |

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

| (D) | | |
|---|---|---|
| *tid* | *iutil* | *rutil* |
| 1 | 5 | 0 |
| 2 | 5 | 0 |
| 3 | 5 | 0 |
| 4 | 5 | 0 |
| 5 | 5 | 0 |
| 6 | 5 | 0 |
| 7 | 5 | 0 |

**Fig. 2.** The constructed UL structures of 1-itemsets from the running example.

Suppose $tid(t)$ denotes the *tid* of transaction $t$, $X.tids$ denotes the *tid* tuples in the utility-list of $X$, and $X'.tids$ that in $X'$, thus:

$\because X \subset X' \Rightarrow X'.tids \subseteq X.tids$.

$$\therefore u(X') = \sum_{id(t)\in X'.tids} u(X',t) \leq \sum_{id(t)\in X'.tids} u(X,t) + rutil(X,t)$$
$$\leq \sum_{id(t)\in X.tids} u(X,t) + rutil(X,t)$$
$$< utilmax(f(X)).$$

$\because f(X') \leq f(X)$, $\therefore \exists$ itemset $Y$ such that $f(Y) \geq f(X) \geq f(X')$ and $u(Y) = utilmax(f(X)) \geq u(X') \Rightarrow X'$ is not a SFUP. □

Based on the provided two lemmas, we can conclude a pruning strategy as follows:.

**Pruning Strategy:** If the summed up value of $(iutil + rutil)$ in the $UL$ structure of $X$ is less than $utilmax(f(X))$, all the extensions of $X$ are not SFUP.

### 4.3. The SKYFUP-D mining algorithm

In the designed SKYFUP-D algorithm, the depth-first search is to find the extensions (supersets) of $X$ in different level (or length). Details of the designed algorithm are described below.

**Algorithm 1:** SKYFUP-D

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

1 **for** *each* $i_j \subseteq T_q \wedge T_q \in D$ **do**
2  calculate $tu(T_q)$;
3  find $twu(i_j)$;
4 sort $i_j$ in *twu*-ascending order;
5 $ULs \leftarrow$ **construct**$(i_j)$;
6 **for** *each* $k$ $(1 \leq k \leq |D|)$ **do**
7  $utilmax(k) := 0$;
8 $SFUPs \leftarrow null$;
9 **D-Search**(*null*, *ULs*, *utilmax*, *SFUPs*);
10 return *SFUPs*;

In the designed Algorithm 1, the transaction-weighted utility of each item is first calculated and the database is then re-organized according to the *twu*-ascending order of each item (Lines 1 to 4). After that, the algorithm is performed to construct the utility-list of each item (Line 5). The *utilmax* values of varied frequencies from 1 to the number of the transactions in $D$ are initially set as 0. This process helps to reduce the search space for later mining process (Lines 6 to 7). The depth-first search approach called D-Search is then performed to explore the search space for mining the SFUPs (Line 8). Details are given below.

**Algorithm 2:** D-Search

**Input**: $P.UL$, the utility of an itemset $P$; *ULs*, the set of utility-list of P's all 1-extensions; $utilmax(k)$, the maximal utility of $k$, $1 \leq k \leq |D|$; $SFUPs$, the set of SFUPs.

1 **for** *each X.ULs* **do**
2  **if** $sum(X.iutil) \geq utilmax(f(X))$ **then**
3  **Judge**(*X*, *utilmax*, *SFUPs*);
4  **if** $sum(X.iutil) + sum(X.rutil) \geq utilmax(f(X))$ **then**
5  $exULs \leftarrow null$;
6  **for** *each Y.ULs after X in ULs* **do**
7  $exULs = exULs +$ **construct**$(P.UL, X, Y)$;
8  **D-Search**(*X*, *exULs*, *utilmax*, *SFUPs*);

Algorithm 2 shows the pseudo-code of the depth-search approach for mining SFUPs. For each utility-list of $X$ in $ULs$ ($X.ULs$), if the sum of all the *iutil* of $X$ is no less than $utilmax(f(X))$, then $X$ may be a SFUP according to Lemma 1 (Lines 2 to 3). After that, the extensions of $X$ is necessary to be determined whether it is required to be explored as the search space for later mining process. Based on the designed Lemma 2, if the summed up value of *iutil* and *rutil* of $X$ is no less than the $utilmax(f(X))$, the search space can thus be explored (Lines 4 to 8). The set of ULs of all 1-extensions of $X$ is recursively processed by intersecting $X.ULs$ and each $Y.ULs$ after $X$ in $ULs$ until no candidates are required to be determined (Lines 6 to 7). The Judge function is described as below.

**Algorithm 3:** Judge

**Input**: $X$, the potential SFUP; $utilmax$, an array of $utilmax(k)$, $1 \leq k \leq |D|$; $SFUPs$, the set of SFUPs.

1 find the first $Y$ in $SFUPs$, $f(Y) > f(X)$;
2 **if** $Y == null$ *or* $u(X) > u(Y)$ **then**
3  insert X to $SFUPs$;
4  **for** *each* $Y$ *in SFUPs,* $f(Y) < f(X)$ **do**
5  **if** $u(X) \geq u(Y)$ **then**
6  remove $Y$ from $SFUPs$;
7  **for** *each* $Y$ *in SFUPs,* $f(Y) == f(X)$ **do**
8  **if** $u(X) > u(Y)$ **then**
9  remove $Y$ from $SFUPs$;
10  **for** $z := f(X)$ **do**
11  **if** $utilmax(z) < u(X)$ **then**
12  $utilmax(z) := u(X)$;
13  $z := z$ - 1;

For the itemset $X$ in Algorithm 3, if its utility is higher than $utilmax(f(X))$, it necessitates to check whether it is a SFUP. It finds the first itemset $Y$ which has higher frequency than $X$ in SFUPs (Line 1).

Since $u(X)$ is no less than $utilmax(f(X))$, $u(X)$ must higher than or equal to $u(Y)$. Since $f(X) < f(Y)$, thus if $u(X) == u(Y)$, $X$ must not be a SFUP since $Y$ dominates $X$. If $Y$ is *null* or $u(X) > u(Y)$, $X$ must be a SFUP; it needs to judge whether an itemset is still a SFUP if its frequency is less than or equal to $X$ after inserting $X$ into SFUPs (Lines 3 to 9). Besides, the $utilmax$ is corresponding updated at the same time (Lines 10 to 13) for later mining process.

### 4.4. SKYFUP-B mining algorithm

The SFUPs can also be discovered by the breath-first search, which is shown in SKYFUP-B algorithm. The pseudo-code of SKYFUP-B is similar as the SKYFUP-D except B-Search approach is used to instead of D-Search approach. The pseudo-code of B-Search is given below.

**Algorithm 4:** B-Search

**Input**: $P.UL$, the utility of an itemset $P$; $ULs$, the set of utility-list of P's all 1-extensions; $utilmax(k)$, the maximal utility of $k$, $1 \le k \le |D|$; $SFUPs$, the set of SFUPs.

```
1 set Q ← null;
2 Scan(null, ULs, utilmax, SFUPs, Q);
3 Extend(utilmax, SFUPs, Q);
4 while Q! = null do
5     while Q! = null ∧ Q.first.visited == false do
6         tmp = Q.out();
7         Scan(tmp.UL, tmp.ULs, utilmax.SFUPs, Q);
8     Extend(utilmax, SFUPs, Q);
```

In the phase for searching SFUPs, it first constructs a queue ($Q$) to store the itemsets which have visited but not yet to be extended and the itemsets which have not visited. The 1-itemsets are then discovered (indicates they are visited but not extended). The *utilmax* of each frequency is then updated, as well as the SFUPs and $Q$ (Line 2). After that, it is necessary to determine whether the 1-itemsets could be extended according to Lemma 2 (Line 3). Thus, some unpromising candidates can be early pruned. If the set of $Q$ is not empty, it means that some itemsets have visited but not yet to be extended or have not visited. It first handles all the unvisited itemsets in the head of $Q$ (Lines 5 to 7). After that, the visited itemsets in the head of $Q$ can be determined to check whether the extension should be explored according to Lemma 2 (Line 8).

**Algorithm 5:** Scan

**Input**: $P.UL$, the utility of an itemset $P$; $ULs$, the set of utility-list of P's all 1-extensions; $utilmax(k)$, the maximal utility of $k$, $1 \le k \le |D|$; Q, the queue of the itemsets for exploration.

```
1 for each X in ULs do
2     if sum(X.iutil) ≥ utilmax(f(X)) then
3         Judge(X, utilmax, SFPUs);
4         X.visited = true;
5         Q.in(X);
```

Algorithm 5 shows the pseudo-code of mining SFUPs from unvisited itemsets in the head of $Q$. For each utility-list of $X$ in $ULs$ ($X.ULs$), if the sum of all the $iutil$ of $X$ is no less than $utilmax(f(X))$, then $X$ may be a SFUP according to Lemma 1 and need a further judgement (Lines 2 to 3). Besides, it means that $X$ has been visited, and should insert it into $Q$ to judge whether its extensions are SFUPs (Lines 4 to 5).

Algorithm 6 shows the pseudo-code for extending visited itemsets in the head of $Q$ and continue to judge whether the extensions of visited itemsets are SFUPs. Set $X$ be the first visited itemset in $Q$, and export it from $Q$ (Line 2). $X$ is the 1-extension of itemset $P$ and $ULs$ keeps the set of utility-list of $P$'s all 1-extensions (Lines 3 to 4). According to Lemma 2, if the sum of $iutil$ and $rutil$ of $X$ is no less than the $utilmax(f(X))$, its extensions may be SFUPs and need to be explored. The set of $ULs$ of

**Algorithm 6:** Extend

**Input**: $utilmax(k)$, the maximal utility of $k$, $1 \le k \le |D|$; $SFUPs$, the set of SFUPs; Q, the queue of the itemsets for exploration.

```
1  while Q! = null ∧ Q.first.visited == true do
2      set X ← Q.out();
3      set P ← X.prefix;
4      set ULs ← the set of utility-list of P's all 1-extensions ;
5      if sum(X.iutil) + sum(X.rutil) ≥ utilmax(f(X)) then
6          exULs ← null;
7          for each Y after X in ULs do
8              exULs= exULs + construct(P.UL, X, Y);
9              construct(P.UL, X, Y).visited = false;
10             Queue.in(construct(P.UL, X, Y));
```

**Table 3**
A sorted quantitative database.

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

all 1-extensions of $X$ is recursively proceed by intersecting $X.ULs$ to $Y.ULs$ until no candidates are required to be determined (Lines 5 to 10).

## 5. An illustrated example

In this section, a database shown in Table 1 and the profit table shown in Table 2 are used as the running example to illustrate the procedure of the designed SFKFUP algorithm. First, the *twu* values of all items in the database are discovered and the results are {$twu(A)$:68, $twu(B)$:66, $twu(C)$:66, $twu(D)$:97, $twu(E)$:60}.

After that, the items in the database are then re-sorted according to their *twu-ascending* and the results are shown in Table 3. In this example, the *utilmax* of each frequency is initially set as 0.

### 5.1. Depth-first search

From the *twu* values of 1-items and Table 3, we can build the *UL* structures for all items shown in Fig. 2. Since $(E)$ has the least $twu$ value among all 1-itemsets, $(E)$ is then first processed. The frequency of $(E)$ is calculated as $f(E)(= 4)$, and the utility of $(E)$ can be also calculated from Fig. 2 as $u(E)(= 20)$. Since the utility-max (*utilmax*) values from 1 to 7 were initially set as 0, thus the utility-max values from 1 to 4 are corresponding updated as $utilmax(1) = utilmax(2) = utilmax(3) = utilmax(4)$ (= 20).

Since the depth-first search is applied in the designed algorithm, the extensions (supersets) of $(E)$ are then determined. The summed up value of *iutil* and *rutil* of $(E)$ in Fig. 2 is calculated as 60, which is larger than the *utilmax*(4)(= 20); the extensions of $(E)$ are thus explored. From Fig. 2, the extensions of $(E)$ can be discovered as $(EB)$, $(EC)$, $(EA)$, and $(ED)$ and their *UL* structures are then respectively built. The $(EB)$ is first checked to decide whether it is a SFUP. In this example, $f(EB)(= 2)$ and $u(EB)(= 18) < utilmax(2)(= 20)$; $(EB)$ could not be a SFUP. The summed up value of *iutil* and *rutil* of $(EB)$ is calculated as 40, which is larger than $utilmax$ (= 20); the extensions of $(EB)$ is then later explored. In this example, the itemset (EBC) is then determined. In this example, $f(EBC)(= 1)$ and $u(EBC)(= 9) < utilmax(1)(= 20)$; $(EBC)$ is not considered as the SFUP. The summed up value of *iutil* and *rutil* of $(EBC)$ is calculated as 18, which is less than $utilmax$ (= 20); it is unnecessary to explore the extensions of $(EBC)$ according to

**Table 4**
Parameters of used datasets.

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

Lemma 2. After that, the itemsets $(EC)$, $(EA)$, and $(ED)$ are processed in the same way until all necessary candidates are traversed. From the given example, the discovered SFUPs are $(BAD)$, $(ED)$ and $(D)$. The search space of the depth-first search of the running example is shown in Fig. 3(a), and the unvisited itemsets are the itemsets pruned by the pruning strategy.

### 5.2. Breath-first search

The designed algorithm can also apply the breath-first search to find the set of SFUPs. In the running example, the itemset $(E)$ is first determined as well. Since $f(E)$ (= 4), the utility-max values from 1 to 4 are respectively updated as $utilmax(1) = utilmax(2) = utilmax(3) = utilmax(4)$ (= 20). The same process is then performed for the other items in the same level, such as $(B)$, $(C)$, $(A)$, and $(D)$. After that, the utility-max values from 1 to 7 are respectively updated as $utilmax(1) = utilmax(2) = utilmax(3) = utilmax(4) = utilmax(5) = utilmax(6) = utilmax(7)$ (= 35). The summed up value of *iutil* and *rutil* of $(E)$ is calculated as 60, which is larger than $utilmax$ (= 35); the extensions of $(E)$ are then explored such as $(EB)$, $(EC)$, $(EA)$, and $(ED)$. The extensions of $(B)$, $(C)$, $(A)$ and $(D)$ are performed in the same way until all necessary candidates are traversed. From the given example, the discovered results are the same as the depth-first search method as $(BAD)$, $(ED)$ and $(D)$. The search space of the breath-first search of the running example is shown in Fig. 3(b), and the unvisited itemsets are the itemsets pruned by the pruning strategy.

## 6. Experimental evaluation

In this section, the designed algorithms are then compared with the state-of-the-art SKYMINE (Goyal et al., 2015) and the SFU-Miner (Pan et al., 2017) algorithms in terms of runtime, memory usage, search space size, and scalability. It is not reasonable to compare the designed algorithms to the traditional top-k methods since those methods can only consider one aspect to find the desired information, and the discovered information cannot obtain more than one factor for revealing better solutions. Thus, only the state-of-the-art SKYMINE (Goyal et al., 2015) and the SFU-Miner (Pan et al., 2017) algorithms are compared to the designed approaches. The algorithms were implemented in Java and experiments were carried out on a personal computer equipped with an Intel Core2 i3-4160 CPU and 4 GB of RAM, running on the 64-bit Microsoft Windows 7 operating system. Four real-world datasets called chess (Anon, 2012), mushroom (Anon, 2012), foodmart (Microsoft, 0000) and retail (Anon, 2012) and one synthetic dataset called T10I4N4KD100K (Agrawal and Srikant, 1994b), were used in the experiments to evaluate the performance of the proposed algorithms. In the experiments, the algorithm is then terminated if the runtime exceeds 12 h or the memory leakage occurs. Parameters and characteristics of these datasets are respectively shown in Tables 4 and 5.

### 6.1. Runtime

The proposed algorithms by breath-first and depth-first searches were compared with the state-of-the-art SKYMINE and SFU-Miner algorithms on four real-world datasets and one synthetic dataset. The results are shown in Table 6. As what we mentioned before, if the runtime exceeds 12 h and the memory leakage occurs, the algorithm is then terminated and it showed '–' in the experiments.

**Table 5**
Characteristics of the used 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 |

**Table 6**
Runtime of the compared algorithms.

| Unit (s) | SKYMINE | SFU-Miner | SKYFUP-D | SKYFUP-B |
|---|---|---|---|---|
| Chess | – | 202.69 | 35.19 | – |
| Mushroom | 202.82 | 10.57 | 2.16 | 1.67 |
| Foodmart | 98.59 | 2.64 | 2.20 | 1.95 |
| Retail | – | 117.57 | 108.51 | 1.24 |
| T10I4N4KD100K | 346.45 | 57.32 | 61.01 | 0.75 |

**Table 7**
Memory usage of the compared algorithms.

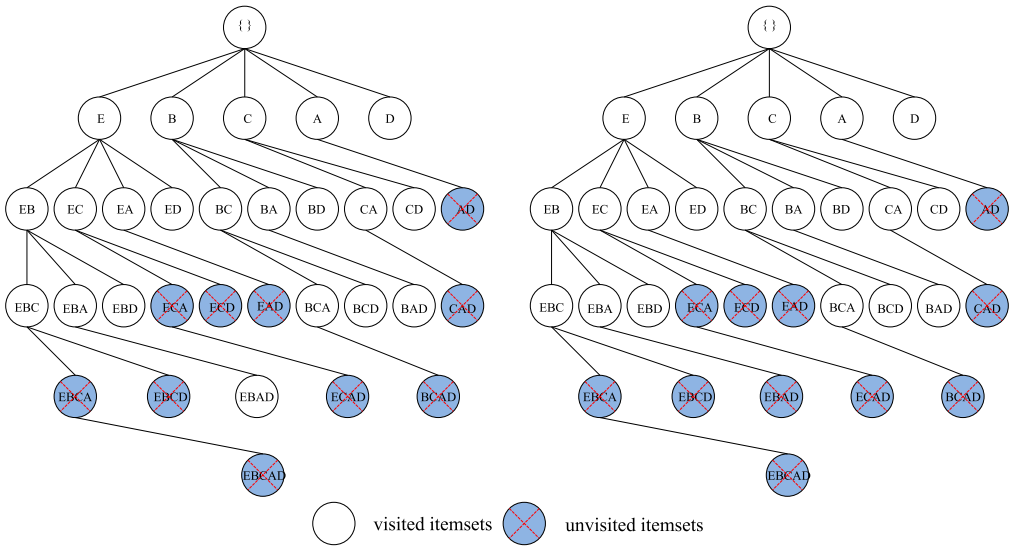
| Unit (MB) | SKYMINE | SFU-Miner | SKYFUP-D | SKYFUP-B |
|---|---|---|---|---|
| Chess | – | 137.00 | 161.84 | – |
| Mushroom | 423.80 | 18.87 | 182.71 | 122.17 |
| Foodmart | 667.46 | 32.13 | 166.9 | 131.19 |
| Retail | – | 143.68 | 143.77 | 147.08 |
| T10I4N4KD100K | 446.69 | 234.30 | 124.28 | 62.58 |

From Table 6, it can be seen that the proposed SKYFUP algorithms outperforms the state-of-the-art SKYMINE and SPF-Miner algorithms and the SKYFUP-B algorithm is generally up to almost two orders of magnitude faster than the SKYMINE algorithm, and up to almost one time order of magnitude faster than the SFU-Miner algorithm for the dense datasets. For the sparse datasets, the SKYFUP-B algorithm is sometimes up to almost three or four orders of magnitude faster than the SKYMINE algorithm and up to almost three orders of magnitude faster than the SFU-Miner algorithm. For example in the mushroom dataset, the runtime of the SKYMINE, the SFU-Miner, the SKYFUP-D and the SKYFUP-B were respectively 202.82, 10.57, 2.16 and 1.67 s. From the above results, we can see that the presented UL structure utilizes the mining performance and the designed algorithms can efficiently reduce the computational cost by the developed pruning strategies. Besides, the proposed algorithms can directly find the actual utility values of the patterns in twice database scans but the SKYMINE algorithm needs to generate the amounts of candidates for finding the skyline points and the SFU-Miner needs more computational cost since it does not adopt the efficient pruning strategies. In addition, the SKYFUP-B algorithm always outperforms the SKYFUP-D algorithm. It is reasonable since the breadth-first search efficiently update the *utilmax* values than that of the depth-first search; more candidates can be early pruned and the search space can be reduced.

### 6.2. Memory usage

In this section, the memory usage of the proposed algorithms under two search methods and the SKYMINE algorithm were compared. Memory usage is measured using Java API and the results are shown in Table 7.

From Table 7, it can be observed that the proposed algorithms with two different searches require less memory than that of the SKYMINE and the SFU-Miner algorithms. For example in mushroom dataset, the SKYMINE, the SFU-Miner, the proposed SKYFUP-D and the SKYFUP-B respectively need 423.80 MB, 18.87, 182.71 MB and 122.17 MB. This is reasonable since the proposed algorithms are based on the UL structure for finding the skyline points, which can be easily performed by the simple join operation. The amounts of candidates can be early pruned by the designed approach but the SKYMINE algorithm needs




(a) depth-first search (b) breadth-first search

**Fig. 3.** Search space of two different methods.

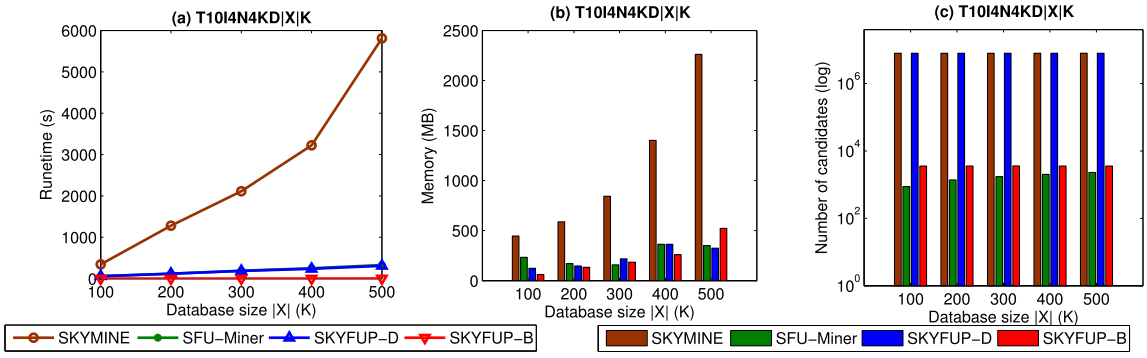



**Fig. 4.** Scalability of the compared algorithms.

memory to store a large of candidates by the pattern-growth mining approach. Although the SFU-Miner has good performance in terms of memory usage, it takes high computational cost in terms of runtime to obtain the SFPUs. For the sparse dataset such as T10I4N4KD100K, the SKYMINE and SFU-Miner need more memory usage than the designed two approaches. Moreover, it can be also found that the SKYFUP-D algorithm generally needs more memory than the SKYFUP-B algorithm in most cases except the chess dataset. Since the chess dataset is very dense, the patterns in the queue for determination by the breath-first search is very large; the memory usage by the breadth-first search is thus worse than the depth-first search in chess dataset.

### 6.3. Search space size

In this subsection, the number of search space for exploring the SFUPs is thus evaluated in different datasets. The results are shown in Table 8.

From Table 8, it can be observed that the SKYMINE algorithm needs a very large search space for mining the SFUPs compared to the designed algorithm SKYFUP-B. For example in the chess dataset, the SKYFUP-B algorithm explores 301 nodes as the search space while the SKYMINE explores 1972 nodes as the search space. The SKYFUP-D almost outperforms the SKYMINE algorithm except the mushroom and T10I4N4KD100K. Although the SFU-Miner needs less search space to obtain the SFPUs compared to the designed SKYFUP algorithms, it lacks of the efficient strategies to quickly determine the SFPUs, which can be found in terms of execution time. Moreover, the SKYFUP-B algorithm always has better results than the SKYFUP-D algorithm. This is reasonable since the breath-first search can easily update the *utilmax* value than that of the depth-first search. Thus, more unpromising candidates can be early pruned and the search space can be greatly reduced.

**Table 8**
The number of search space of the compared algorithms.

| Unit (size) | SFPUs | SKYMINE | SFU-Miner | SKYFUP-D | SKYFUP-B |
|---|---|---|---|---|---|
| Chess | 33 | – | 3,076 | 4,049,053 | – |
| Mushroom | 4 | 12,916 | 1,384 | 58,567 | 301 |
| Foodmart | 6 | 1,214,466 | 61 | 1,214,394 | 1,106,028 |
| Retail | 2 | – | 832 | 220,420,554 | 16,471 |
| T10I4N4KD100K | 4 | 7,994,015 | 879 | 11,795,886 | 3,569 |

### 6.4. Scalability

The scalability of the compared algorithms is conducted on the synthetic dataset T10I4N4KD|X|K, where the dataset size has been varied from 100K to 500K transactions, increments 100K each time. The results are then shown in Fig. 4.

From Fig. 4, it can be observed that the proposed algorithms have good scalability when the dataset size is varied, both in terms of runtime, memory consumption, and number of search space, compared with

the state-of-the-art SKYMINE and the SFU-Miner algorithms. From the results of Fig. 4(a), the SKYMINE requires most time to obtain the SFPUs and the SFU-Miner and the SKYFUP-D require the similar time to obtain the SFPUs. We also can observe that the SKYFUP-B needs much fewer time to obtain the solutions. From Fig. 4(b), it can be observed that the amount of memory usage increases along with the size of the dataset. We also can see that the SKYMINE needs a high amount memory usage to obtain the SFPUs and the SKYFUP-B needs less memory usage than the SFU-Miner and the SKYFUP-D in most cases, for example, when the dataset sizes are varied set from 100k to 400k, increments 100k each time. An interesting observation is that the runtime and memory consumption of the proposed algorithms and the SFU-Miner steadily increase but the SKYMINE sharply increases along with the increasing of dataset size. For example, the SKYFUP-D algorithm was almost one or two orders of magnitude faster than the SKYMINE and has the similar results as the SFU-Miner. However, the SKYFUP-B algorithm was almost three or four orders of magnitude faster than the SKYMINE algorithm when the dataset size was increased from 100K to 500K, as shown in Fig. 4(a). Moreover, the proposed algorithms do not generate any candidates based on the UL structure and the SKYFUP-D always has the similar results as the SFU-Miner. However, the SKYMINE algorithm generates many candidates based on the UP-tree structure. From Fig. 4(c), we can observe that the SKYFUP-D generates nearly the same number of candidates as the SKYMINE, and the SKYFUP-B has a bit more candidates to obtain the SFPUs compared to the SFU-Miner. From the observed results of the scalability experiments, it can be concluded that the proposed algorithms have good scalability and haves better performance in terms of runtime, memory consumption, and number of search space compared to the traditional SKYMINE and the SFU-Miner algorithms.

## 7. Conclusion and future work

In the past, many approaches have been proposed to respectively mine frequent itemsets or high utility itemsets from datasets. However, to the best of our knowledge, very few researches considered both frequency and utility factors together for decision-making. In this paper, two algorithms called SKYFUP-D and SKYFUP-B are respectively proposed to mine the itemset that not be dominated by others under the frequency and utility measures. The designed algorithms rely on the utility-list structure for mining SFUPs without any candidates generation, which solves the problem for generating massive candidates. Besides, the designed algorithms are unnecessary to set the non-trivial minimum support or utility thresholds but the returned skyline points are concisely meaningful for decision-making. Substantial experiments were conducted on both real-life and synthetic datasets to assess the performance of the proposed algorithms in terms of runtime, memory consumption, number of search space and scalability.

Since only few works were designed to address the problem for mining SFUPs, several extensions such as mining SFUPs from the stream data, from the uncertain data or from the dynamic database can be further explored. Besides, it is also necessary to design more efficient structure to keep the necessary information for mining the SFUPs.

## References


Afrati, F.N., Koutris, P., Suciu, D., Ullman, J.D., 2015. Parallel skyline queries. Theory Comput. Syst. 57 (4), 1008–1037.

Agrawal, R., Imielinsky, T., Swami, A., 1993. Mining association rules between sets of items in large databases. In: International Conference on Management of Data, pp. 207–216–499.

Agrawal, R., Srikant, R., 1994a. Fast algorithm for mining association rules. In: International Conference on Very Large Data Bases, pp. 487–499.

Agrawal, R., Srikant, R., 1994b. Quest synthetic data generator. http://www.Almaden.ibm.com/cs/quest/syndata.html.

Ahmed, C.F., Tanbeer, S.K., Jeong, B.S., Le, Y.K., 2009. Efficient tree structures for high utility pattern mining in incremental databases. IEEE Trans. Knowl. Data Eng. 21 (12), 1708–1721.

Anon, 2012. Frequent itemset mining dataset repository, http://fimi.ua.ac.be/data/.

Borzsonyi, S., Kossmann, D., Stocker, K., 2001. The skyline operator. In: International Conference on Data Engineering, pp. 421–430.

Chan, C.Y., Jagadish, H.V., Tan, K.L., Tung, A.K.H., Zhang, Z., 2006. Finding k-dominant skylines in high dimensional space. In: ACM SIGMOD International Conference on Management of Data, pp. 503–514.

Chan, R., Yang, Q., Shen, Y.D., 2003. Mining high utility itemsets. In: IEEE International Conference on Data Mining, pp. 19–26.

Chomicki, J., Godfrey, P., Gryz, J., Liang, D., 2003. Skyline with presorting. In: International Conference on Data Engineering, pp. 717–720.

Fournier-Viger, P., Lin, J.C.W., Kiran, R.T., Koh, Y.S., Thomas, R., 2017. A survey of sequential pattern mining. Data Sci. Pattern Recognit. 1 (1), 54–77.

Fournier-Viger, P., Wu, C.W., Tseng, V.S., 2012. Mining top- k association rules. In: The Canadian Conference on Artificial Intelligence, pp. 61–73.

Fournier-Viger, P., Wu, C.W., Zida, S., Tseng, V.S., 2014. FHM: faster high-utility itemset mining using estimated utility co-occurrence pruning. Found. Intell. Syst. 8502, 83–92.

Gan, W., Lin, J.C.W., Fournier-Viger, P., Chao, H.C., Hong, T.P., Fujita, H., Fujita, Hamido, 2018b. A survey of incremental high-utility itemset mining. Wiley Interdisciplinary Rev. Data Min. Knowl. Discov. 8 (2), 1–23.

Gan, W., Lin, J.C.W., Fournier-Viger, P., Chao, H.C., Tseng, V.S., Yu, P.S., 2018a. A survey of utility-oriented pattern mining. https://arxiv.org/abs/1805.10511.

Goyal, V., Sureka, A., Patel, D., 2015. Efficient skyline itemsets mining. In: The International C* Conference on Computer Science & Software Engineering, pp. 119–124.

Han, J., Pei, J., Yin, Y., 2000. Mining frequent patterns without candidate generation. In: ACM SIGKDD International Conference on Management of Data, pp. 1–12.

Kossmann, D., Ramsak, F., Rost, S., 2002. Shooting stars in the sky: an online algorithm for skyline queries. In: International Conference on Very Large Data Bases, pp. 275–286.

Kung, H., Luccio, F., Preparata, F., 2005. On finding the maxima of a set of vectors. J. ACM 22 (4), 502–513.

Lin, C.W., Hong, T.P., Lu, W.H., 2011. An effective tree structure for mining high utility itemsets. Expert Syst. Appl. 38 (6), 7419–7424.

Lin, X., Yuan, Y., Zhang, Q., Zhang, Y., 2007. Selecting stars: the k most representative skyline operator. In: International Conference on Data Engineering, pp. 86–95.

Liu, Y., Liao, W.K., Choudhary, A., 2005. A two-phase algorithm for fast discovery of high utility itemsets. Lecture Notes in Comput. Sci. 3518, 689–695.

Liu, J., Pan, Y., Wang, K., 2002. Mining frequent itemsets by opportunistic projection. In: ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 229–238.

Liu, M., Qu, J., 2012. Mining high utility itemsets without candidate generation. In: ACM International Conference on Information and Knowledge Management, pp. 55–64.

Microsoft, 0000. Example database foodmart of Microsoft analysis services, http://msdn.microsoft.com/en-us/library/aa217032(SQL.80).aspx.

Pan, J.S., Lin, J.C.W., Yang, L., Fournier-Viger, P., Hong, T.P., 2017. Efficiently mining of skyline frequent-utility patterns. Intell. Data Anal. 21 (6), 1407–1423.

Papadias, D., Tao, Y., Seeger, B., 2005. Progressive skyline computation in database systems. ACM Trans. Database Syst. 30 (1), 41–82.

Park, J.S., Chen, M.S., Yu, P.S., 1995. An effective hash based algorithm for mining association rules. In: ACM SIGMOD International Conference on Management of Data, pp. 175–186.

Pei, J., Han, J., Lu, H., 2001. H-mine: hyper-structure mining of frequent patterns in large databases. In: IEEE International Conference on Data Mining, pp. 441–448.

Pei, J., Jin, W., Ester, M., Tao, Y., 2005. Catching the best views of skyline: a semantic approach based on decisive subspaces. In: International Conference on Very Large Data Bases, pp. 253–264.

Podpecan, V., Lavrac, N., Kononenko, I., 2007. A fast algorithm for mining utility-frequent itemsets. In: International workshop on Constraint-based Mining and Learning, pp. 9–20.

Savasere, A., Omiecinski, E., Navathe, S., 1995. An efficient algorithm for mining association rules in large databases. In: International Conference on Very Large Databases, pp. 432–444.

Tan, K.L., Eng, P.K., Ooi, B.C., 2001. Efficient progressive skyline computation. In: International Conference on Very Large Data Bases, pp. 201–310.

Tseng, V.S., Shie, B.E., Wu, C.W., Yu, P.S., 2012. Efficient algorithms for mining high utility itemsets from transactional databases. IEEE Trans. Knowl. Data Eng. 25 (8), 1772–1786.

Tseng, V.S., Wu, C.W., Fournier-Viger, P., Yu, P.S., 2015. Efficient algorithms for mining top-k high utility itemsets. IEEE Trans. Knowl. Data Eng. 28 (1), 54–67.

Tseng, V.S., Wu, C.W., Shie, B.E., Yu, P.S., 2010. UP-growth: An efficient algorithm for high utility itemset mining. In: ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 253–262.

Wu, C.W., Shie, B.E., Tseng, V.S., Yu, P.S., 2012. Mining top- k high utility itemsets. In: ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 78–86.

Yao, H., Hamilton, H.J., 2006. Mining itemset utilities from transaction databases. Data Knowl. Eng. 59 (3), 603–626.

Yao, H., Hamilton, H.J., Butz, C.J., 2004. A foundational approach to mining itemset utilities from databases. In: SIAM International Conference on Data Mining, pp. 211–225.

Yao, H., Hamilton, H.J., Geng, L., 2006. A unified framework for utility-based measures for mining itemsets. In: ACM SIGKDD International Conference on Utility-Based Data Mining, 28–37.

Yeh, J.S., Li, Y.C., Chang, C.C., 2007. Two-phase algorithms for a novel utility-frequent mining model. In: International Conference on Emerging Technologies in Knowledge Discovery and Data Mining, pp. 433–444.

Yen, S.J., Lee, Y.S., 2007. Mining high utility quantitative association rules. Lecture Notes in Comput. Sci. 4654, 283–292.

Zaki, M.J., Parthasarathy, S., ogihara, M., Li, W., 1997. New algorithm for fast discovery of association rules. In: International Conference on Knowledge Discovery and Data Mining, pp. 283–286.