Computer Science and Information Systems 20(3):1085–1108 https://doi.org/10.2298/CSIS220615040W

The Effective Skyline Quantify-utility Patterns Mining Algorithm with Pruning Strategies

Jimmy Ming-Tai Wu[1], Ranran Li[1], Pi-Chung Hsu[2], and Mu-En Wu[3,⋆]

[1] College of Computer Science and Engineering, Shandong University of Science and Technology
Shandong, China
wmt@wmt35.idv.tw
734181156@qq.com
[2] Department of Information Management, Shu-Te University
Kaohsiung, Taiwan
pichung@stu.edu.tw
[3] Department of Information and Finance Management, National Taipei University of Technology
Taipei, Taiwan
mnasia1@gmail.com

Abstract. Frequent itemset mining and high-utility itemset mining have been widely applied to the extraction of useful information from databases. However, with the proliferation of the Internet of Things, smart devices are generating vast amounts of data daily, and studies focusing on individual dimensions are increasingly unable to support decision-making. Hence, the concept of a skyline query considering frequency and utility (which returns a set of points that are not dominated by other points) was introduced. However, in most cases, firms are concerned about not only the frequency of purchases but also quantities. The skyline quantity-utility pattern (SQUP) considers both the quantity and utility of items. This paper proposes two algorithms, FSKYQUP-Miner and FSKYQUP, to efficiently mine SQUPs. The algorithms are based on the utility-quantity list structure and include an effective pruning strategy which calculates the minimum utility of SQUPs after one scan of the database and prunes undesired items in advance, which greatly reduces the number of concatenation operations. Furthermore, this paper proposes an array structure superior to utilmax for storing the maximum utility of quantities, which further improves the efficiency of pruning. Extensive comparison experiments on different datasets show that the proposed algorithms find all SQUPs accurately and efficiently.

Keywords: Internet of Things, skyline quantity-utility patterns (SQUPs), utility-quantity list, minimum utility of SQUPs (MUSQ), quantity maximum utility of the array (QMUA).

1. Introduction

The Internet of Things (IoT) has resulted in the daily generation of massive amounts of data, making the extraction of valuable information a significant challenge. Data mining techniques, also known as knowledge discovery from databases (KDD) [2,18,30,47], can be applied in this endeavor. Association rule mining (ARM) [3,4,5] and frequent item-set

⋆ Corresponding author

mining (FIM) [16,17,27,48] are traditional methods for processing data. ARM typically finds not only frequent itemset (FI) patterns based on a user-defined minimum support threshold (*minsup*) but also correlations or causal structures between different item sets based on a minimum confidence threshold (*minconf*). ARM and FIM are widely applied in fields such as news recommendation, weather correlation analysis, precision marketing, and price prediction.

Both FIM and ARM count how many times a commodity appears in a transaction by measuring if a specific commodity or a combination of commodities is present. This means that other essential factors, such as the profit of the commodity or the number of purchases, are not considered. In practice, these factors are often more important to the user. In order to further satisfy the needs of users, a concept called high-utility itemset mining (HUIM) has been proposed, it is gradually becoming the focus of research in the field of big data [6,14,44,45]. In a large shopping mall, for example, the number of luxury bags sold in a single day is much lower than the number of daily necessities. However, the profits generated by luxury bags might be higher than those of daily necessities. Yao *et al*. [45] proposed finding high-utility item sets (HUIs) by considering the number of items and the profit per unit of items. In FIM, if an item set $\{AB\}$ is frequent, then any subset of this item set, such as $\{A\}$ or $\{B\}$ is frequent; however, in HUIM, if an item set $\{AB\}$ is an HUI, its subset $\{A\}$ or $\{B\}$ is not necessarily an HUI. Thus, HUIM does not satisfy the downward closure property. If there are n items, then 2^n-1 combinations are generated, which requires a large search space in order to determine whether this set of items is a conforming HUI. To solve this difficulty, Liu *et al*. [26] proposed a new model called TWU, in which the utility also satisfies the downward closure property, which greatly narrows the search space. Subsequently, several scholars have researched and successfully proposed new algorithms and effective pruning strategies [7,34,40,43].

To achieve information extraction, these algorithms require the user to set a threshold, which determines the final quality of the results. If the value is too high, much of the useful information will be ignored. If the value is too small, much of the extracted information will be redundant. Setting a suitable parameter is also time-consuming and inefficient for the user. To address this challenge, the concept of Top-k [12,37] was proposed, i.e., the user can extract the top k most essential pieces of information from the database by setting a parameter k. Although this approach significantly shortens the decision-making process, information is only extracted from a single aspect. FIM can help users to find goods that are frequently purchased, and HUIM can help users to find goods that can earn high profits; however, it is important to firms to know what goods are frequently purchased and generate high profits. Therefore, Goyal *et al*. [15] proposed an algorithm to find the frequent-utility skyline (SFU), which is a set of points measuring frequency and utility that are not dominated by each other. Considering that the quantity of items purchased by users is also a concern in real life, Wu *et al*. [42] subsequently designed the skyline quantity-utility pattern (SQUP) model to include the factor of quantity and proposed two algorithms based on UQL structure: SQU-Miner and SKYQUP. However, because these two algorithms generate numerous candidate sets, they create a vast search space.

With the widespread adoption of the IoT, intelligent decision support systems (IDSSs) have evolved into powerful tools for extracting useful information from large amounts of data. This paper proposes a smart supermarket model to demonstrate the application of the proposed algorithm (see Fig. 1). Touchable smart electronic screens, gravity sensors, and image sensors are all included in the proposed smart shopping cart. The electronic screens summarize the list of products purchased and calculate the total number of items purchased. These electronic screens send information back to the supermarket's data center, and the supermarket can use the proposed algorithm to find non-dominated points and extract valuable patterns. Based on these, the supermarket can design effective marketing strategies.

The main contributions of this paper are as follows:

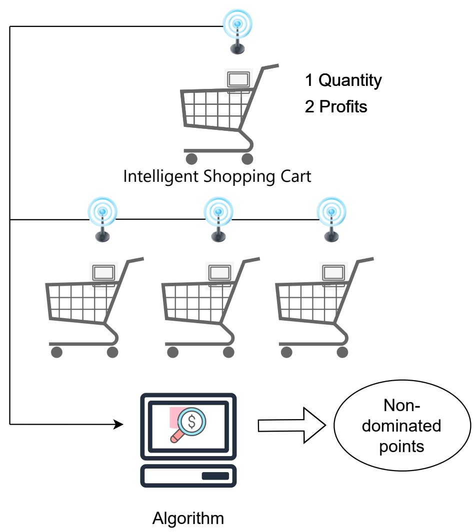

Fig. 1. Skyline model framework in smart supermarket

1. This paper presents two efficient UQL structure-based algorithms for mining SQUPs (FSKYQUP-Miner and FSKYQUP), both of which are depth-first search-based algorithms that do not require user-defined thresholds.
2. The maximum utility of the quantity is stored in a *QMUA* array, and based on this array, an efficient pruning strategy is proposed to prune undesired candidates and their extended sets.
3. The minimum utility of the SQUPs (*MUSQ*) is found in order to eliminate undesired individual items and all their extended sets in the initial stage of the algorithm using the TWU property. This dramatically narrows the search space of the algorithm.
4. Extensive experiments were conducted on real-world and synthetic datasets, the results of which demonstrate the efficacy of the proposed algorithm compared to existing approaches.

The remainder of this thesis is organized as follows. In Sect. 2, we review the current research on HUIM and skyline queries. Sect. 3 presents relevant formulas and definitions while Sect. 4 details the proposed algorithm, including the proposed pruning strategy and the pseudo-code of the algorithm. Sect. 5 presents the experimental comparison results, and a summary and directions for future research are presented in Sect. 6.

2. Related Work

In this section, we briefly review research on high-utility itemset mining and skyline queries.

2.1. High-utility itemset mining

FIM algorithms in general include level-wise and pattern-growth algorithms. Apriori [5] was the first algorithm proposed for the former, and it satisfies the DC property. However, it does generate massive candidate sets and necessitates several database scans to compute these candidate sets. To address this issue, a new FP-Growth algorithm based on pattern growth [17] is proposed, which is based on the compact data structure FP-Tree. It only scans the database once and does not generate any candidate sets for recursively mining FIs from the database. While other algorithms for investigating FIs have been proposed in recent years, all are single-minded and can only compute the frequency of item sets while ignoring critical metrics such as quantity, weight, and utility.

With its focus on utility, HUIM has been widely studied as an important tool for data mining. HUIM computes the revenue generated by a commodity or combination of commodities and compares it to a minimum revenue parameter specified by the user; if it is greater than this parameter, this item is placed in an HUI. Because HUIM lacks DC like the Apriori algorithm, Liu *et al*. [26] proposed upper bound TWU in order to find more comprehensive HUIs. The TWU-based model, however, necessitates multiple database scans and a vast search space. Subsequently, Lin *et al*. [22] proposed a new structure called a high-utility pattern (HUP) tree based on the FP-Tree to improve the quality of mining performance. However, precisely because the algorithm is based on the FP-Tree, a large portion of memory is needed to store the generated intermediate nodes. Therefore, Tseng *et al*. proposed a new UP-Tree structure to maintain similarity with the FP-Tree structure and proposed two algorithms, UP-Growth [38] and UP-Growth+ [36], to efficiently mine HUIs by reducing the number of database scans. These tree-structure algorithms nevertheless generate a large number of candidate item sets. Liu *et al*. [25] created a new utility list (UL) structure based on the TWU model and proposed the HUI-Miner algorithm. This structure does not require multiple scans of the database and does not generate a large number of candidate sets. The list concatenation operation makes mining HUIs simple, efficient, and complete. Further HUIM extensions have subsequently been proposed [13,24], including the top-k algorithm [12,39], which mines the top k eligible item sets in the database to overcome the necessity of setting a threshold value. Modifications have also been proposed [20,41,49] to reduce the algorithm runtime by improving pruning strategies and designing better data structures.

2.2. The previous hybrid approach

The works reviewed above focus on a single factor, which is inconvenient for decision-making. Yeh *et al*. [46] thus combined utility and frequency in the FUP model; however, in this approach, a threshold must still be set by the user. Podpecan *et al*. [31] proposed a novel algorithm to increase mining efficiency that also requires user-defined parameters. Goyal *et al*. [15] then proposed SKYMINE, which does not require the user to set any parameters. This algorithm is based on the well-known UP-Tree structure and returns a set of points for decision-making that is not dominated by any other points. However, due to the limitations of its data structure, the algorithm generates numerous candidate sets and is thus inefficient. Pan *et al*. [28] proposed an efficient utility list structure-based SFU-Miner algorithm to reduce the number of candidate sets. Lin *et al*. [23] proposed two algorithms based on the UL structure, called SKYFUP-D and SKYFUP-B algorithms, which are two typical algorithms based on DFS and BFS search. Although the application of list structure has dramatically improved mining efficiency, researchers continue to search for more effective pruning strategies. Song *et al*. [33] proposed SFUI-UF, which deletes undesired item sets from the database in the initial stages of the algorithm and thus considerably shortens runtime. Song *et al*. [32] also proposed cross-entropy-based mining algorithm SFU-CE to improve mining efficiency. These algorithms all consider the utility and frequency of items but neglect the fact that in practice, the quantity of items purchased is still the primary concern of users. Wu *et al*. [42] were the first to suggest considering utility and quantity, proposing the SQUP model and two new algorithms to mine SQUPs.

2.3. The skyline concept

Mining SFUPs from a database is, in general, a multi-objective optimization case that considers frequency and utility and returns a set of points as a solution. That is, subsets $\{a_1, a_2, ..., a_m\}$ (holding information valuable to the user) are found within a large set of databases *D*. These subsets are not dominated by other points in at least one dimension. If, for example, there exists a point b_n which is better than a_n in all dimensions, then a_n is dominated by b_n and will eventually return to b_n as the decision point instead of a_n. This skyline result is highly relevant to real-world scenarios. For instance, parents may consider house price and distance from schools when choosing a suitable residence. Generally, house prices close to schools will be higher than those far from schools; therefore, parents look for distances and prices that are relatively suitable. In Fig. 2, the *x*-coordinate represents the distance to the school, with larger values representing longer distances; the *y*-coordinate represents house prices, with larger values representing higher prices; and the buildings in the figure represent houses available for rent. The houses $\{g, c, l\}$ in the figure are the skyline points because these points are not dominated by other points in the dimensions of distance and price; therefore, these houses represent the best choices.

Kung *et al*. [21] introduced the skyline concept in 2005, using a “partitioning” strategy to find skyline points. Borzsonyi *et al*. [8] were the first to combine skylines and databases, proposing an algorithm based on block nested loops, which gained wide attention. Chomicki *et al*. [10] improved this block nested loop algorithm using a specific tuple order in the window to improve the performance. Tan *et al*. [35] proposed two algorithms, Bitmap and Index, which output skyline points step by step, unlike the usual algorithms that need to traverse the dataset at least once to return the first point. Kossmann *et al*. [19] proposed an NN algorithm based on nearest-neighbor search and used a form of “partitioning” to compute skyline queries. Papadias *et al*. [29] proposed the branch-and-bound skyline (BBS) algorithm, which is also based on nearest-neighbor search and has the characteristics of I/O so that it can be applied to various asymptotic operations. These explorations of skyline computation have been widely discussed [1,9].

Traditional algorithms FIM and HUIM consider only one factor, while skyline algorithms return non-dominated points based on multiple factors. This paper proposes a list-based FSKYQUP-Miner and FSKYQUP algorithm to mine SQUPs using the utility quantity list structure for the join operation. The preparatory knowledge and problem statement of skyline quantity utility pattern mining (SQUPM) are presented in the following section.

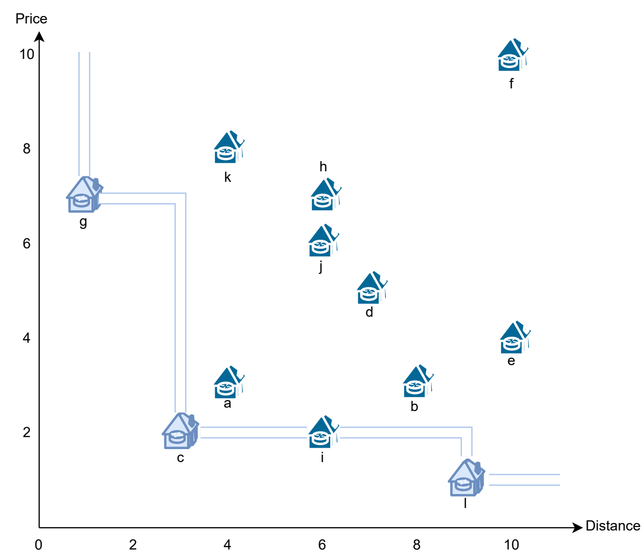

Fig. 2. Example of skyline points

3. Preliminary Knowledge and Problem Statement

3.1. Preliminaries

Assuming that $D = \{T_1, T_2, ..., T_n\}$ is a transaction database with n transactions, $I = \{i_1, i_2, ..., i_m\}$ is a set of m distinct items in the database. In D, each transaction $T_q \in D$ is a subset of I containing a number of items and their purchased quantities $q(i_j, T_q)$, along with a unique identifier called *TID*. Additionally, a profit table called *ptable* = $\{pr_1, pr_2, ..., pr_m\}$, where pr_j is the per-unit profit (profit) generated by each item i_j (i.e., good). An itemset $X = \{i_1, i_2, ..., i_k\}$ is a set of k distinct items, where k is the length of the k-itemset. If $X \subseteq T_q$, then the set of items X is said to occur in transaction T_q. In this paper, our running example is shown in Table 1, which is a database consists of 7 transactions, and in Table 2 the profit corresponding to each item in the running example is given.

Table 1. Original transaction database DB in the running instance

T_{ID}	Item and its quantity	Transaction utility
T_1	B:2,D:2,E:3	23
T_2	A:2,B:3	8
T_3	B:2,C:3,D:4,E:1	33
T_4	B:2,D:2	14
T_5	A:1,D:2,E:2	17
T_6	C:3,E:2	12
T_7	A:2,B:1,C:1,D:1,E:2	17

Table 2. Unit profit table of the item in the running example

Item	Profit
A	1
B	2
C	2
D	5
E	3

Definition 1. *In the transaction T_q, the quantity of the itemset X to be purchased is denoted as $q(X, T_q)$, a mathematical definition of which is as follows:*

$$q(X, T_q) = min\{q(Y)|Y \subseteq X \wedge X \in T_q \wedge Y \in T_q\}. \tag{1}$$

It is obtained from T_2 in Table 1 that *q*(*A*) = 2*, q*(*B*) = 3*,* so the quantity of the itemset (*AB*) is the smallest one, which is 2.

Definition 2. *The utility of an item i_j in a transaction T_q is called as $u(i_j, T_q)$, a mathematical definition of which is as follows:*

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

It is obtained from T_2 in Table 1 that the utility of the item $\{A\}$ can be computed as $u(A, T_2) = q(A, T_2) \times pr(A) = 2 \times 1 = 2$, the utility of the item $\{B\}$ can be computed as $u(B, T_2) = q(B, T_2) \times pr(B) = 3 \times 2 = 6$.

Definition 3. *The utility of an itemset X in a transaction T_q is called as $u(X, T_q)$, a mathematical definition of which is as follows:*

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

It is obtained from T_2 in Table 1 that the utility of the itemset $\{AB\}$ can be computed as $u(AB, T_2) = u(A, T_2) + u(B, T_2) = 2 + 6 = 8$.

Definition 4. *The utility of itemset X in a transaction database D is called as $u(X)$, a mathematical definition of which is as follows:*

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

It is obtained from Table 1 that the utility of itemset $\{B\}$ in database D can be computed as $u(B) = u(B, T_1) + u(B, T_2) + u(B, T_3) + u(B, T_4) + u(B, T_7) = 4 + 6 + 4 + 4 + 2 = 20$, $u(BD) = u(BD, T_1) + u(BD, T_3) + u(BD, T_4) + u(BD, T_7) = 14 + 24 + 14 + 7 = 59$.

Definition 5. *The utility of a transaction in a transaction database D is called as $tu(T_q)$, a mathematical definition of which is as follows:*

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

It is obtained from Table 1 that there are 3 items in T_1, which are B, D and E, so $tu(T_1) = u(B, T_1) + u(D, T_1) + u(E, T_1) = 4 + 10 + 9 = 23$. The transaction utility of other transactions in the running example is shown on the right side of Table 1, $tu(T_2)$ = 8, $tu(T_3) = 33$, $tu(T_4)$ = 14, $tu(T_5)$ = 17, $tu(T_6)$ = 12, $tu(T_7)$ = 17.

Definition 6. *The transaction-weighted utility of an itemset X in a transaction database D is called as twu(X), a mathematical definition of which is as follows:*

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

It is obtained from Table 1 that item $\{A\}$ appears in T_2, T_5, T_7, so the twu of the itemset $\{A\}$ is called as $twu(A) = tu(T_2) + tu(T_5) + tu(T_7)$ = 42.

For the sake of taking both quantity and utility into account, the concept of skyline quantity-utility pattern mining (SQUPM) is listed below:

Definition 7. *For itemset X and itemset Y, if $q(X) \geqslant q(Y)$ and $u(X) > u(Y)$ or $q(X) > q(Y)$ and $u(X) \geqslant u(Y)$, then the itemset X governs Y and it is represented as $X \succ Y$.*

It is obtained from Table 1 that $q(A)$ = 5, $q(B)$ = 10, and $u(A)$ = 5, $u(B)$ = 20. It can be said that the item $\{B\} \succ \{A\}$ because $u(B) > u(A)$ and $q(B) > q(A)$.

Definition 8. *When considering two-dimensional factor quantity and utility, an itemset is said to be SQUPM if it behaves as if it is not governs by other itemsets in the database.*

3.2. Problem statement

Via the above definition, the problem of mining SQUPM can be formally defined as finding all sets of ungoverned points, namely SQUPs, from a quantitative database *D*.

For the running example in Table 1, the utility and quantity of $\{ED\}$ are computed as 69 and 6, the utility and quantity of $\{BD\}$ are computed as 59 and 7, and the utility and quantity of $\{D\}$ are computed as 55 and 11. Since any one of these three points cannot dominate the others, the sets $\{ED\}$, $\{BD\}$, and $\{D\}$ are eventually returned as skyline points.

4. Proposed Algorithms to Mine SQUPs

This paper proposes two depth-first search-based algorithms, FSKYQUP-Miner and FSKYQUP. This section consists of five subsections. In the first subsection, the utility-quantity-list structure is introduced, which is the basis of the algorithm proposed in this section. The proposed new structure is introduced in the second subsection. The third subsection introduces the pruning strategy used in the proposed algorithm. The fourth subsection describes the proposed algorithm in detail. The pseudo-code used will also be shown in this section, and the last part will be a step-by-step detailed mining process with the running example in Table 1.

4.1. Utility-quantity-list structure

In database D, calculate the TWU of each item and sort the TWU in ascending order using the $\rhd$ represent. Create a utility-quantity-list (UQL) [42] structure for each item, which is a quadruplet containing (*tid*, *quantity*, *utility*, *remaining utility*). Where *tid* represents the transaction ID containing this item, quantity (abbreviated as *quan*) is to calculate the quantity of purchases of this item in the *tid*, utility (abbreviated as *iutil*) is to calculate the utility of this item in this *tid*, and remaining utility (abbreviated as *rutil*) is to calculate the sum of the utilities of the items appearing in this item in a *tid* after sorting by $\rhd$.

Definition 9. *The mathematical definition of rutil is as follows:*

$$rutil(X) = \sum_{i_j \subseteq T_q/X} iutil(i_j, T_q). \tag{7}$$

Assume in Table 1 that $\rhd$ indicates that the items are sorted in ascending order based on the transaction-weighted utility of each item; then the sorted items are $A \rhd C \rhd B \rhd E \rhd D$, and in transaction T_1, the items that appear after item B after the sorting are item E and item D. As a result, in the transaction, $rutil = iutil(E, T_1) + iutil(D, T_1) = 9+10 = 19$.

4.2. Quantity maximum utility of the array (QMUA)

In this section, two efficient array structures for storing the maximum utility of quantities are proposed to record and update the maximum utility of itemsets, which largely reduces the search space for mining SQUPs.

Definition 10. (*Quantity Maximum Utility of the Array*) *Define q_{max} to be the maximum quantity of all 1-itemsets in the database D.*

If the quantity q(X) of an itemset X $(1 \leqslant q(X) \leqslant q_{max})$ is equal to the original parameter i, then the QMUA structure will be defined as follows:

$$QMUA1(i) = max\{u(X) \mid q(X) = i\}. \tag{8}$$

If the quantity of itemset X, $q(X)$ $(1 \leqslant q(X) \leqslant q_{max})$, is greater than or equal to the original parameter i, then the $QMUA$ structure will be defined as follows:

$$QMUA2(i) = max\{u(X) \mid q(X) \geqslant i\}. \tag{9}$$

Among them, unlike utilmax, the size of utilmax is set to $\mid D \mid$ and the size of *QMUA* is set to $q_{max} + 1$. *QMUA*1 and *QMUA*2 update in different ways; *QMUA*1 only updates the utility of the set of items whose quantity is equal to i, whereas *QMUA*2 updates the utility of all sets of items whose quantity is greater than or equal to i.

Definition 11. *Computing the quantity of an itemset X in the database D as q(X) = q. An itemset X is called a potential SQUP (PSQUP) if none of the other itemsets with quantity q has a utility greater than u(X).*

Theorem 1. *If an itemset X is not a PSQUP, then it cannot be an SQUP. That is, SQUP ⊆ PSQUP.*

Proof. For $\forall X \notin PSQUPs$, there $\exists$ an itemset Y that makes $q(Y) = q(X) \wedge u(Y) > u(X)$. According to Definition 7, it is known that Y dominates X. So $\forall X \notin SQUPs$.

This maximum utility array structure of quantity is used in the algorithms proposed in this paper to update the maximum utility of storing an equal quantity of itemsets using the *QMUA* structure, which can greatly reduce the space needed to search during the mining of SQUPs. In addition, the update method *QMUA*1 corresponds to the FSKYQUP-Miner algorithm proposed in this paper, and the update method *QMUA*2 corresponds to the FSKYQUP algorithm. For simplicity, the two update methods of *QMUA* are directly distinguished by the algorithm names in the following text.

4.3. Pruning strategies

In this portion, a pruning strategy for the initial phase of two algorithms and two pruning strategies in the mining phase will be presented.

Definition 12. *(minimum utility of SQUPs) In the original database D, the minimum utility of SQUPs is defined as the maximum utility of the largest quantity of the 1-itemset, a mathematical definition of which is as follows:*

$$MUSQ = max\{u(X)|q(X) = q_{max}\}. \tag{10}$$

Where X is a 1-itemset in database D, and q_{max} is the maximum quantity of all 1-itemsets computed. Taking the running example, the quantity of item D in database D is $q(D)$ = 2 + 4 + 2 + 2 + 1 = 11, so q_{max} = 11, and since $u(D)$ = 55, $MUSQ$ = 55.

Theorem 2. *An itemset X is a 1-itemset in the database. If the TWU of X is less than MUSQ, then this itemset X and all its extensions are not SQUPs.*

Proof. Assume Y is another 1-itemset in the database, and with $q(Y) = q_{max}$, $u(Y) = MUSQ$.
$\therefore u(X) \leqslant TWU(X) < MUSQ = u(Y) \wedge q(X) \leqslant q_{max} = q(Y)$.
Y dominates X.
$\therefore X \notin SQUPs$.
Assume that eX is an arbitrary extended set of items containing itemset X.
$\therefore u(eX) \leqslant TWU(X) < MUSQ = u(Y) \wedge q(eX) \leqslant q(X) \leqslant q_{max} = q(Y)$.
$\therefore eX$ is dominated by Y.
$\therefore$ Any extension set of X is not $SQUPs$.

Therefore, according to Theorem 2, the set of items with a TWU smaller than *MUSQ* can be directly pruned at the beginning of the algorithm, which greatly reduces the number of candidate sets. Furthermore, it is essential for the efficiency of the algorithm that the value of *MUSQ* be assigned to the *QMUA* array as the initial value.

Theorem 3. *An itemset X is not a SQUP if the sum of the iutil of the itemset X is less than the QMUA value corresponding to q(X).*

Proof. Suppose there exists an itemset Y with $q(Y) \geqslant q(X)$, $u(Y) = QMUA(q(X))$
$\because X.sumiutil < QMUA(q(X)) = u(Y) \Rightarrow u(X) < u(Y)$
since $q(X) \leqslant q(Y)$
$\therefore Y$ dominates $X \Rightarrow X \notin SQUPs$.

According to the Theorem 3, it is possible to prune those terms whose sum of utilities of the itemset is less than *QMUA*, and these items are not SQUPs.

Theorem 4. *Any extension eX of X is not a SQUPs if the sum of iutil and rutil of the extension eX of the itemset X is less than the QMUA value corresponding to q(X).*

Proof. Assume that eX is an arbitrary extended set of items containing itemset X.
$\therefore$ For $\forall$ transaction T, it is possible to obtain:
$eX \subseteq T \Rightarrow (eX - X) = (eX/X) \Rightarrow (eX/X) \subseteq (T/X)$
$\therefore u(eX, T) = u(X, T) + u(eX - X, T)$
$= u(X, T) + u(eX/X, T)$
$= u(X, T) + \sum\limits_{i_j \in eX/X} u(i_j, T)$
$\leqslant u(X, T) + \sum\limits_{i_j \in T/X} u(i_j, T)$
$= u(X, T) + rutil(X, T)$
$\because q(X) \geqslant q(eX)$
$\therefore eX.tids \leqslant X.tids$
$\therefore u(eX) = \sum\limits_{tid = eX.tids} u(eX, T)$
$\leqslant \sum\limits_{tid \in eX.tids} u(X, T) + rutil(X, T)$
$\leqslant \sum\limits_{tid \in X.tids} u(X, T) + rutil(X, T) < QMUA(q(X))$.
$\therefore \exists$ an itemset Y that makes $q(Y) \geqslant q(X) \geqslant q(eX)$, $u(Y) = QMUA(q(X)) \geqslant u(eX)$
$\therefore Y$ dominates $eX \Rightarrow eX \notin SQUPs$.

According to the sum of $iutil$ and $rutil$ of the itemset in Theorem 4, it can be determined whether the extension of the itemset is PSQUPs or not. If the sum is less than *QMUA*, then the extension of this item is not SQUPs and the extension of this item can be cut to reduce the search space.

4.4. The proposed algorithm

This paper proposes two UQL structure-based algorithms, FSKYQUP-Miner and FSKYQUP, to find SQUPs quickly and efficiently. Both algorithms are based on depth-first search, and the itemsets are ordered among themselves. In addition, the difference between the two algorithms is the different update methods, i.e., Algorithm 3 and Algorithm 4. The two algorithms and their related pseudo-code will be shown in the following.

Algorithm 1 is the pseudo-code of the proposed algorithms. Firstly, the database is scanned for the first time and the TWU of single items, the maximum quantity q_{max} and *MUSQ* are calculated (line 1 of the algorithm). According to Theorem 2, if the TWU of the item is less than *MUSQ*, then this item i_j is deleted from the database and the database is pruned in the initial stage of this algorithm (lines 2–4 of the algorithm). Lines 5–6 sort the items in ascending order of TWU and reorganize the database. This loop creates a UQL structure for each item in the reorganized database (lines 7–11). Then the *QMUA* is initialized to *MUSQ*, the maximum utility for the largest quantity of itemsets (lines 12–14 of the algorithm). It is worth noting that although the FSKYQUP-Miner algorithm and the FSKYQUP algorithm are updated in different ways, the initialization is the same. The **Search** function is then called to find all SQUPs (shown in detail in Algorithm 2). A set of SQUPs has finally been returned.

Algorithm 1 FSKYQUP-Miner/FSKYQUP algorithm

Require:
  Original database D; profit table.
Ensure:
  A set of SQUPs.
1: Scan the database D and calculate the TWU of the item i_j, $q_{max}, MUSQ$;
2: **if** $TWU(i_j) < MUSQ$ **then**
3:   Delete i_j from original database D;
4: **end if**
5: Sorting items i_j by TWU in ascending order;
6: Reorganization database;
7: **for** each $T_q \in$ *re-D* **do**
8:   **for** each $i_j \in T_q$ **do**
9:     Create $i_j.UQLs$;
10:   **end for**
11: **end for**
12: **for** *i* = 1 to q_{max} **do**
13:   *QMUA*(*i*) = *MUSQ*;
14: **end for**
15: set *SQUPs* = null;
16: **Search** (*null*, *UQLs*, *QMUA*, *SQUPs*);
17: return *SQUPs*;

Algorithm 2 mines SQUPs based on depth-first search. For each itemset *X* belonging to the UQL (where UQL refers to the UQL corresponding to each extension of the prefix), if the sum of the utilities of the itemset *X* is greater than or equal to the *QMUA* of *q*(*X*), the itemset *X* may be SQUPs according to Theorem 3, and the **Judge** function is called to determine whether it is the final SQUP (lines 3–5). The subsequent lines 6–9 are to determine whether the extensions of the itemset *X* are psqups, and if the sum of *iutil* and *rutil* of *X* is greater than or equal to the *QMUA* of *q*(*X*), its extension *eX* is psqup.

According to Theorem 4, the extended UQL is established. Line 10 of the algorithm is a recursive call process until lines 7-8 no longer yield candidates.

Algorithm 2 Search

Require:
PUQL, UQL of the current prefix; *UQLs*, the UQL corresponding to each extension of the prefix; *QMUA*; *SQUPs*.

1: **for** i = 0 to *UQLs.size* **do**
2:     X = *UQLs.get*(i);
3:     **if** $X.sumiutil \geqslant QMUA[q(X)]$ **then**
4:         **Judge** (X, *QMUA*, *SQUPs*);
5:     **end if**
6:     **if** $X.sumiutil + X.sumrutil \geqslant QMUA[q(X)]$ **then**
7:         **for** each $Y \lhd X$ **do**
8:             $eXUQLs \leftarrow Create(PUQL, X, Y)$;
9:         **end for**
10:         **Search** (X, *eXUQLs*, *QMUA*, *SQUPs*);
11:     **end if**
12: **end for**

Algorithm 3 and Algorithm 4 are pseudo-codes based on the FSKYQUP-Miner algorithm and FSKYQUP algorithm, respectively, to determine whether the itemset X is SQUPs. The difference between the two algorithms lies in the different update methods, which are explained in detail by Algorithm 3 as an example. If the *sumiutil* of an itemset X exceeds *QMUA*[$q(X)$], it is necessary to investigate whether this itemset is an SQUP. In the first line of the algorithm, if Y is the first itemset in the SQUP set whose quantity is greater than X, i.e., $q(Y)$ is greater than $q(X)$, then the itemset X is an SQUP only when Y is equal to the empty set or when the utility of the itemset X is greater than the utility of the itemset Y. Then, insert X into the set of SQUPs. Otherwise, the itemset Y will dominate the itemset X and X must not be an SQUP. Then, update the value of *QMUA* (line 4 of the algorithm). Next, determine whether, after inserting X, the set of SQUPs with a quantity less than X is an SQUP (lines 5-7 of the algorithm).

4.5. Illustrative example

Using the FSKYQUP-Miner algorithm as an example, the database used in the example is displayed in Table 1, and the profit table is displayed in Table 2. After the first scan of database D, it is calculated that $q(D) = q_{max} = 11$ and *MUSQ* = 55. The TWU of each item in the database is $\{A$: 42, B: 95, C: 62, D: 104, E: 102$\}$. Since TWU(A) = 42 < *MUSQ* = 55, according to Theorem 2, item A and all its extended itemsets are not SQUPs, and therefore, item A is removed from the database. The remaining items, after sorting in ascending order by TWU are $C \rhd B \rhd E \rhd D$. According to this order, the original database will be reorganized, and the reorganized database is shown in Table 3.

Moreover, a UQL structure is created for each item as shown in Table 4. After initialization, *QMUA*[1] to *QMUA*[11] are assigned a value of 55.

Algorithm 3 Judge-FSKYQUP-Miner

Require:
X, the PSQUP; QMUA; SQUPs.
find the first $Y \in SQUPs$, and $q(Y) > q(X)$;
if $Y == null$ or $u(X) > u(Y)$ **then**
  SQUPs $\leftarrow X$;
  $QMUA[q(X)] = X.sumiutil$;
  for each itemset $Y \in$ *SQUPs* **do**
    if $q(X) = q(Y) \wedge u(X) > u(Y)$ or $q(X) > q(Y) \wedge u(X) \geqslant u(Y)$ **then**
      delete Y from SQUPs;
    end if
  end for
end if

Algorithm 4 Judge-FSKYQUP

Require:
X, the PSQUP; QMUA; SQUPs.
find the first $Y \in SQUPs$, and $q(Y) > q(X)$;
if $Y == null$ or $u(X) > u(Y)$ **then**
  SQUPs $\leftarrow X$;
  for $n = q(X)$ down to 1 **do**
    if $X.sumiutil > QMUA[n]$ **then**
      $QMUA[n] = X.sumiutil$;
    end if
  end for
  for each itemset $Y \in$ *SQUPs* **do**
    if $q(X) = q(Y) \wedge u(X) > u(Y)$ or $q(X) > q(Y) \wedge u(X) \geqslant u(Y)$ **then**
      delete Y from SQUPs;
    end if
  end for
end if

Firstly, starting from C, the UQL of C gives $q(C) = 7$, $iutil(C) = 14 <$ *QMUA*[7] = 55, so C is not a SQUP, and since $iutil(C) + rutil(C) = 60 >$ *QMUA*[7], consider the extensions of C. The items that appear following C after sorting, and are connected to C at the beginning and end, form the extensions of C, which are CB, CE, and CD. Establish UQL for these items. Next, explore CB, Since $q(CB) = 3$, $iutil(CB) = 14 <$ *QMUA*[3], and $iutil(CB) + rutil(CB) = 48$, it is obvious that it is less than *QMUA*[3], so CB and its extensions are not SQUPs. Since the algorithm is based on depth-first search, CE is checked next. According to Table 4, $q(CE) = 4$, $iutil(CE) = 29 <$ *QMUA*[4] = 55, similarly, $iutil(CE) + rutil(CE) = 54 <$ *QMUA*[4], CE and its extensions are also not SQUPs. Next check $CD, q(CD) = 4$, $iutil(CD) = 33 <$ *QMUA*[4], and $iutil(CD) + rutil(CD) = 33 <$ *QMUA*[4], so CD and its extensions are not SQUPs. Follow the same steps to check B. Finally, the discovered candidate sets are {*BED*, *BD*, *ED*, *D*}, and all skyline quantity utility itemsets found are shown in Table 5. The final updated *QMUA* of FSKYQUP-Miner algorithm is {55, 55, 55, 63, 55, 69, 59, 55, 55, 55, 55} while the final updated *QMUA* of FSKYQUP algorithm is {69, 69, 69, 69, 69, 69, 59, 55, 55, 55, 55}

Table 3. Reorganization database in the running instance

T_{ID}	Item and its quantity
T_1	B:2,E:3,D:2
T_2	B:3
T_3	C:3,B:2,E:1,D:4
T_4	B:2,D:2
T_5	E:2,D:2
T_6	C:3,E:2
T_7	C:1,B:1,E:2,D:1

respectively. The *QMUA* array is obviously updated faster in the FSKYQUP algorithm than in the FSKYQUP-Miner algorithm. Coincidentally, within the example of this paper, the search spaces of proposed two algorithms are the same, as shown in Fig. 3.

Table 4. The utility-quantity-list structures of 1-items

(a) *C*

t_{id}	$quan$	$iutil$	$rutil$
3	3	6	27
6	3	6	6
7	1	2	13

(b) *B*

t_{id}	$quan$	$iutil$	$rutil$
1	2	4	19
2	3	6	0
3	2	4	23
4	2	4	10
7	1	2	11

(c) *E*

t_{id}	$quan$	$iutil$	$rutil$
1	3	9	10
3	1	3	20
5	2	6	10
6	2	6	0
7	2	6	5

(d) *D*

t_{id}	$quan$	$iutil$	$rutil$
1	2	10	0
3	4	20	0
4	2	10	0
5	2	10	0
7	1	5	0

Table 5. Excavated SQUPs

SQUPs	quantity	utility
ED	6	69
BD	7	59
D	11	55

5. Experimental Evaluation

The FSKYQUP algorithm and the FSKYQUP-Miner algorithm proposed in this paper are compared with the SKYQUP algorithm and the SQU-Miner algorithm [42], two of the most advanced algorithms for mining SQUPs, in terms of runtime, memory consumption, the number of search itemsets, the resulting candidate sets, and the scalability of the algorithms. The experiments were conducted on a computer with an Intel (R) Core (TM) i3-8100 CPU @ 3.60 GHZ and 16 GB of RAM. The algorithms were written in Java and run on the idea compiler. To evaluate the algorithms' performance in many aspects,

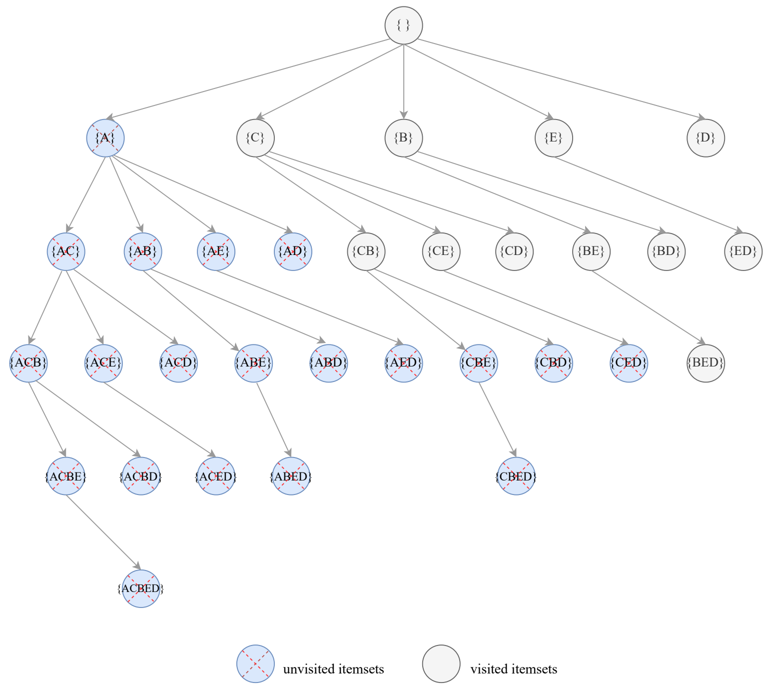

Fig. 3. Search space of proposed algorithms

the experiment was conducted on six different datasets, including five real-world datasets and one synthetic dataset. The following five real-world datasets were downloaded from SPMF [11]: namely Chess, Mushroom, Retail, Foodmart, and Ecommerce. A synthetic dataset T25I10D10K was generated using the utility quantity generator, obeying a Gaussian distribution. The parameters, such as the number of items, are shown in Table 6.

Table 6. Features of the datasets

Dataset	#Trans	#Items	Avg.Trans.Len	Max.Trans.Len	Type
Chess	3196	76	37	37	dense
Mushroom	8124	119	23	23	dense
Retail	88162	16470	10	76	sparse
Foodmart	4141	1559	4.42	14	sparse
Ecommerce	14975	3468	11.64	29	sparse
T25I10D10K	9976	929	24.77	63	dense

Table 6 details the following six characteristics of the six datasets: name of dataset, total number of transactions, number of items, average length of transactions, maximum length of transactions, and type of dataset (sparse or dense).

Runtime The runtimes of the proposed algorithms as well as the state-of-the-art SQU-Miner algorithm and SKYQUP algorithm for each of the datasets are shown in Fig. 4. The number of SQUPs mined for each dataset is shown in Table 7.

Table 7. The number of SQUPs

Dataset	#SQUPs
Chess	38
Mushroom	5
Retail	2
Foodmart	2
Ecommerce	1
T25I10D10K	1

In Fig. 4, generally speaking, the runtime of the FSKYQUP algorithm is shorter than that of the SKYQUP algorithm, and the runtime of the FSKYQUP-Miner algorithm is shorter than that of the SQU-Miner algorithm. In general, the runtimes of the proposed algorithms are shorter than that of SKYQUP and SQU-Miner. The FSKYQUP is better than the FSKYQUP-Miner, although the FSKYQUP-Miner is 0.02 seconds faster than the FSKYQUP on the Foodmart dataset. This is because the updating methods of $QMUA[q]$ differ. FSKYQUP updates based on the utilities of all item sets whose quantity is greater than or equal to q. Meanwhile, FSKYQUP-Miner only updates the utilities of item sets whose quantity is equal to q. Obviously, the FSKYQUP is more efficient at updating and produces fewer candidate item sets. For the dataset Chess, FSKYQUP is superior to the other three algorithms. The FSKYQUP-Miner runs for longer than the SKYQUP algorithm, but for shorter than the SQU-Miner due to the pruning strategy proposed in this paper. For the dataset Retail, the FSKYQUP and the FSKYQUP-Miner are 40 times faster than the SKYQUP and the SQU-Miner. This is because the Retail dataset is sparse, and in general, the items are not as closely related to each other as in a compact dataset. The *QMUA* proposed in this paper is initialized based on the value of *MUSQ*, which means that the utility of most of the item sets does not reach the value for updating. As fewer candidates are generated, the runtime is shorter.

Memory We compared the memory usage of the proposed algorithms with that of the SQU-Miner algorithm and SKYQUP algorithm on each dataset. The experimental results are plotted in Fig. 5.

Fig. 5 shows that except for Mushroom and Foodmart, the proposed algorithms used less memory in mining SQUPs. In particular, the FSKYQUP and FSKYQUP-Miner on the Ecommerce and synthetic dataset T25I10D10K used roughly the same amount of memory, which is nearly 15 times less than the other two datasets. This is due to the efficient pruning strategy proposed in this paper, which narrows the search space. On the Foodmart dataset, the memory usage of the proposed algorithms is more than the existing algorithms, which is attributable to the creation of a list by the proposed algorithms for the storage of undesired candidates. The FSKYQUP-Miner uses the least memory on the Mushroom dataset. For the other two dense-type datasets, the FSKYQUP-Miner saves slightly more memory than the FSKYQUP.

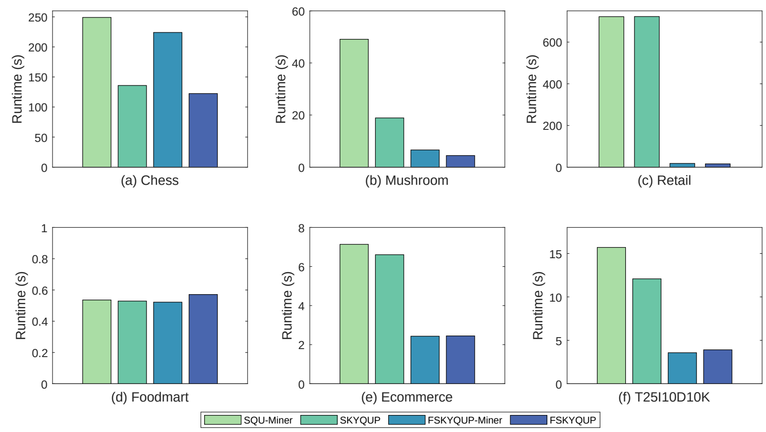

Fig. 4. Runtime on different datasets

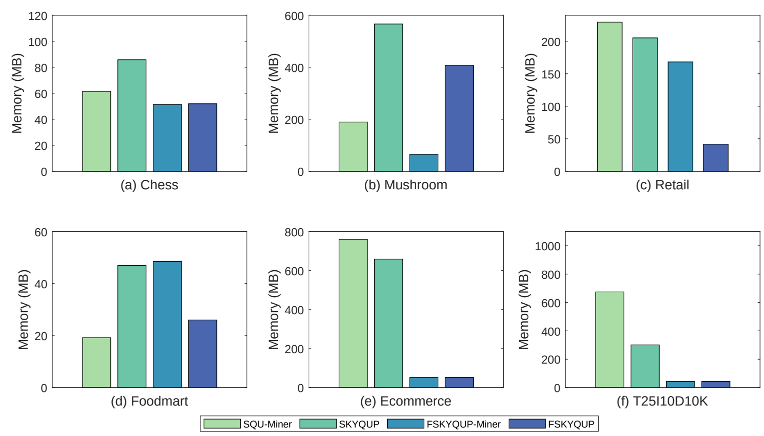

Fig. 5. Memory on different datasets

Search space We evaluated the size of the search space of the four algorithms for different datasets and plotted the results in Fig. 6.

Fig. 6 shows that FSKYQUP and FSKYQUP-Miner require less search space than the other two algorithms, which is due to the efficient pruning strategy proposed in this paper. The FSKYQUP requires the least search space, regardless of whether the dataset is sparse or dense. It is worth noting that on the Retail dataset, the number of search nodes of the SQU-Miner, SKYQUP, FSKYQUP-Miner, and FSKYQUP is respectively 26,803,198,632, 981,229,210, 71, and 71. The difference between the SQU-Miner and the proposed algorithms is eight orders of magnitude. On the Ecommerce dataset, the FSKYQUP and FSKYQUP-Miner are nearly 290 times worse than the other two algorithms in terms of search space. Similarly, on the T25I10D10K dataset, the FSKYQUP and FSKYQUP-Miner are nearly 230 times worse than the other two algorithms in terms of search space. On the datasets Retail, Ecommerce, and T25I10D10K, the FSKYQUP-

Miner requires the same search space as the FSKYQUP algorithm. These results indicate that the weaker the correlation between items in the dataset, the smaller the required search space.

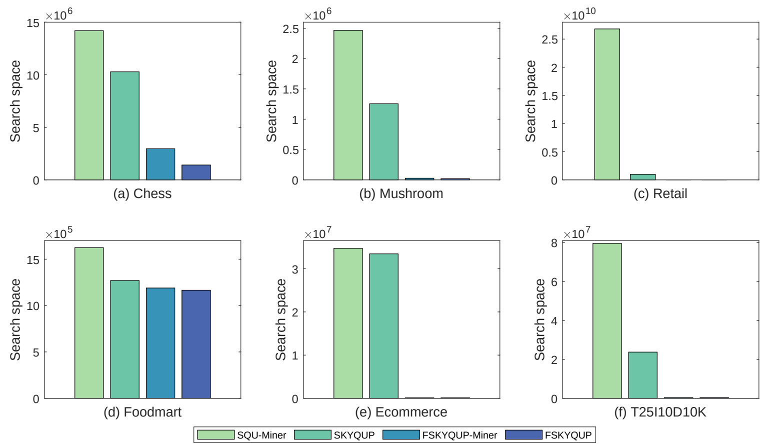

Fig. 6. Search space on different datasets

Candidate We evaluated the number of candidate item sets generated by the four algorithms for each dataset and plotted the results in Fig. 7.

Fig. 7 shows that the number of candidate sets generated by the proposed algorithms is smaller than the other two algorithms for all datasets except the Chess and Foodmart datasets. The FSKYQUP generated the least number of candidate sets for all datasets. For example, for the Retail dataset, the number of candidates generated by the four algorithms is respectively 1,472, 254, 7, and 4. The FSKYQUP generates 368 times fewer candidates than the SQU-Miner. For the reasons described in the first part of this section, the number of candidate sets for Chess and Foodmart generated by the FSKYQUP-Miner is larger than that of the SKYQUP but smaller than that of the SQU-Miner algorithm.

Scalability We conducted scalability experiments on the synthetic dataset, where the transactions of the dataset are set to 100k, 200k, 300k, 400k, and 500k. The performance is compared on each of these datasets in four aspects: runtime, memory usage, search space size, and the number of generated candidate sets. The experimental results are shown in Fig. 8.

The proposed algorithms compare favorably with the state-of-the-art SQU-Miner and SKYQUP algorithms in terms of runtime, memory usage, the size of the search space, and the number of candidate sets generated as the dataset increases. Fig. 8(a) compares the execution times of the four algorithms across the five synthetic data sets. Running the SQU-Miner algorithm takes a long time, and the runtime becomes longer when there are more datasets. The proposed algorithms have similar runtimes and good scalability,

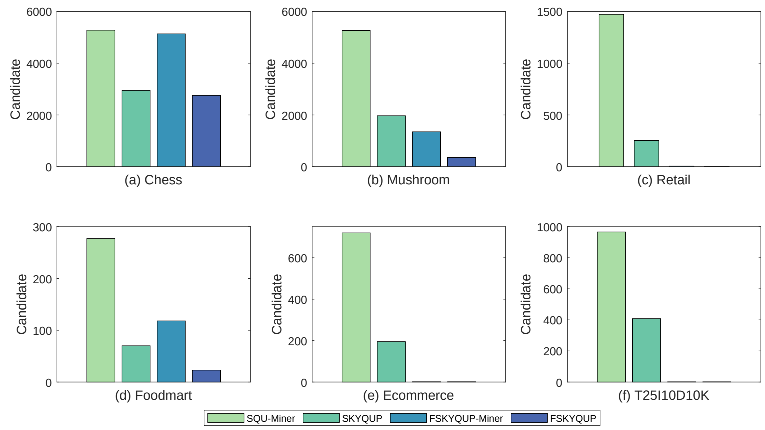

Fig. 7. Candidate on different datasets

as the runtime grows gradually as the dataset increases. Fig. 8(b) displays the memory usage of the four algorithms for the five synthetic datasets. The largest memory consumer is SQU-Miner, while FSKYQUP is the smallest. As the dataset increases, the proposed algorithms have good scalability in terms of memory usage. The search space required to run the four algorithms on different-sized datasets is depicted in Fig. 8(c). It is clear from the figure that the SQU-Miner requires a vast search space, the SKYQUP requires a smaller but still large search space, and the FSKYQUP and FSKYQUP-Miner require the smallest search spaces. Fig. 8(d) depicts the number of candidate sets generated for each dataset: the proposed algorithms generate the least candidate sets, followed by the SKYQUP, while the SQU-Miner generates the most candidate sets. These results indicate that the proposed algorithms offer good scalability in terms of runtime, memory usage, search space, and the number of candidate sets.

6. Conclusion

With the advent of the information age, relying solely on the support of FIs and HUIs is no longer good enough to support decision-making, so people prefer to take into account both the frequency and utility of the work. In contrast, quantity also plays a crucial role in the decision-making process. This paper proposes two methods that do not require a user-defined threshold: FSKYQUP-Miner and FSKYQUP. Both of these approaches are based on UQL and obtain a set of uncontrolled nodes. We also propose a more effective pruning method which eliminates undesired candidates in the initial stage of the algorithm, thus greatly narrowing the search scope. Extensive experiments on real-world and synthetic datasets verified that the proposed methods scale well in terms of runtime, memory usage, search space, and the number of candidate sets. These results indicate that the proposed algorithms are well-suited to supermarket applications. As big data continues to advance, in the future, it would be fruitful to explore SQUPs with other architectures, such as the MapReduce or Spark framework. The proposed algorithms would also benefit from

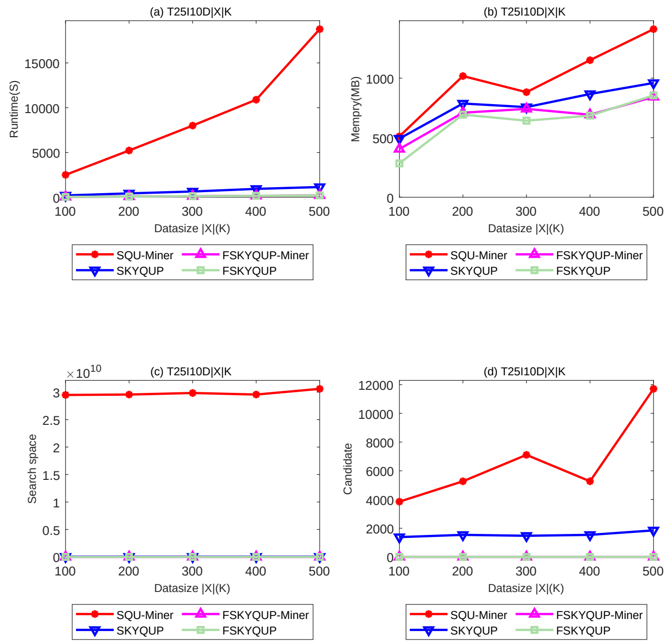

Fig. 8. Scalability on different datasets

additional pruning strategies to simplify the structure and thus mine SQUPs even more effectively.

Acknowledgments. This research is supported by Shandong Provincial Natural Science Foundation (ZR201911150391).

References

1. Afrati, F.N., Koutris, P., Suciu, D., Ullman, J.D.: Parallel skyline queries. Theory of Computing Systems 57(4), 1008–1037 (2015)
2. Agrawal, R., Imielinski, T., Swami, A.: Database mining: A performance perspective. IEEE Transactions on Knowledge and Data Engineering 5(6), 914–925 (1993)
3. Agrawal, R., Imieliński, T., Swami, A.: Mining association rules between sets of items in large databases. In: Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data. pp. 207–216 (1993)
4. Agrawal, R., Mannila, H., Srikant, R., Toivonen, H., Verkamo, A.I., et al.: Fast discovery of association rules. Advances in Knowledge Discovery and Data Mining 12(1), 307–328 (1996)
5. Agrawal, R., Srikant, R., et al.: Fast algorithms for mining association rules. In: Proceedings of the 20th International Conference on Very Large Databases. vol. 1215, pp. 487–499. Citeseer (1994)
6. Ahmed, C.F., Tanbeer, S.K., Jeong, B.S., Lee, Y.K.: Efficient tree structures for high utility pattern mining in incremental databases. IEEE Transactions on Knowledge and Data Engineering 21(12), 1708–1721 (2009)

7. Ahmed, U., Lin, J.C.W., Srivastava, G., Yasin, R., Djenouri, Y.: An evolutionary model to mine high expected utility patterns from uncertain databases. IEEE Transactions on Emerging Topics in Computational Intelligence 5(1), 19–28 (2020)
8. Borzsony, S., Kossmann, D., Stocker, K.: The skyline operator. In: Proceedings 17th International Conference on Data Engineering. pp. 421–430. IEEE (2001)
9. Chan, C.Y., Jagadish, H., Tan, K.L., Tung, A.K., Zhang, Z.: Finding k-dominant skylines in high dimensional space. In: Proceedings of the 2006 ACM SIGMOD International Conference on Management of Data. pp. 503–514 (2006)
10. Chomicki, J., Godfrey, P., Gryz, J., Liang, D.: Skyline with presorting. In: ICDE. vol. 3, pp. 717–719 (2003)
11. Fournier-Viger, P., Lin, J.C.W., Gomariz, A., Gueniche, T., Soltani, A., Deng, Z., Lam, H.T.: The spmf open-source data mining library version 2. In: Joint European Conference on Machine Learning and Knowledge Discovery in Databases. pp. 36–40. Springer (2016)
12. Fournier-Viger, P., Wu, C.W., Tseng, V.S.: Mining top-k association rules. In: Canadian Conference on Artificial Intelligence. pp. 61–73. Springer (2012)
13. Fournier-Viger, P., Wu, C.W., Zida, S., Tseng, V.S.: Fhm: Faster high-utility itemset mining using estimated utility co-occurrence pruning. In: International Symposium on Methodologies for Intelligent Systems. pp. 83–92. Springer (2014)
14. Gan, W., Lin, J.C.W., Fournier-Viger, P., Chao, H.C., Hong, T.P., Fujita, H.: A survey of incremental high-utility itemset mining. Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery 8(2), e1242 (2018)
15. Goyal, V., Sureka, A., Patel, D.: Efficient skyline itemsets mining. In: Proceedings of the Eighth International C* Conference on Computer Science & Software Engineering. pp. 119–124 (2015)
16. Grahne, G., Zhu, J.: Fast algorithms for frequent itemset mining using fp-trees. IEEE Transactions on Knowledge and Data Engineering 17(10), 1347–1362 (2005)
17. Han, J., Pei, J., Yin, Y.: Mining frequent patterns without candidate generation. ACM Sigmod Record 29(2), 1–12 (2000)
18. Imielinski, T., Mannila, H.: A database perspective on knowledge discovery. Communications of the ACM 39(11), 58–64 (1996)
19. Kossmann, D., Ramsak, F., Rost, S.: Shooting stars in the sky: An online algorithm for skyline queries. In: VLDB'02: Proceedings of the 28th International Conference on Very Large Databases. pp. 275–286. Elsevier (2002)
20. Krishnamoorthy, S.: Pruning strategies for mining high utility itemsets. Expert Systems with Applications 42(5), 2371–2381 (2015)
21. Kung, H.T., Luccio, F., Preparata, F.P.: On finding the maxima of a set of vectors. Journal of the ACM (JACM) 22(4), 469–476 (1975)
22. Lin, C.W., Hong, T.P., Lu, W.H.: An effective tree structure for mining high utility itemsets. Expert Systems with Applications 38(6), 7419–7424 (2011)
23. Lin, J.C.W., Yang, L., Fournier-Viger, P., Hong, T.P.: Mining of skyline patterns by considering both frequent and utility constraints. Engineering Applications of Artificial Intelligence 77, 229–238 (2019)
24. Liu, J., Wang, K., Fung, B.C.: Direct discovery of high utility itemsets without candidate generation. In: 2012 IEEE 12th International Conference on Data Mining. pp. 984–989. IEEE (2012)
25. Liu, M., Qu, J.: Mining high utility itemsets without candidate generation. In: Proceedings of the 21st ACM International Conference on Information and Knowledge Management. pp. 55–64 (2012)
26. Liu, Y., Liao, W.k., Choudhary, A.: A two-phase algorithm for fast discovery of high utility itemsets. In: Pacific-Asia Conference on Knowledge Discovery and Data Mining. pp. 689–695. Springer (2005)
27. Luna, J.M., Fournier-Viger, P., Ventura, S.: Frequent itemset mining: A 25 years review. Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery 9(6), e1329 (2019)

28. Pan, J.S., Lin, J.C.W., Yang, L., Fournier-Viger, P., Hong, T.P.: Efficiently mining of skyline frequent-utility patterns. Intelligent Data Analysis 21(6), 1407–1423 (2017)
29. Papadias, D., Tao, Y., Fu, G., Seeger, B.: Progressive skyline computation in database systems. ACM Transactions on Database Systems (TODS) 30(1), 41–82 (2005)
30. Park, J.S., Chen, M.S., Yu, P.S.: An effective hash-based algorithm for mining association rules. Acm Sigmod Record 24(2), 175–186 (1995)
31. Podpecan, V., Lavrac, N., Kononenko, I.: A fast algorithm for mining utility-frequent itemsets. Constraint-Based Mining and Learning p. 9 (2007)
32. Song, W., Zheng, C.: Sfu-ce: Skyline frequent-utility itemset discovery using the cross-entropy method. In: Intelligent Data Engineering and Automated Learning–IDEAL 2021: 22nd International Conference, IDEAL 2021, Manchester, UK, November 25–27, 2021, Proceedings 22. pp. 354–366. Springer (2021)
33. Song, W., Zheng, C., Fournier-Viger, P.: Mining skyline frequent-utility itemsets with utility filtering. In: Pacific Rim International Conference on Artificial Intelligence. pp. 411–424. Springer (2021)
34. Srivastava, G., Lin, J.C.W., Pirouz, M., Li, Y., Yun, U.: A pre-large weighted-fusion system of sensed high-utility patterns. IEEE Sensors Journal 21(14), 15626–15634 (2020)
35. Tan, K.L., Eng, P.K., Ooi, B.C., et al.: Efficient progressive skyline computation. In: VLDB. vol. 1, pp. 301–310 (2001)
36. Tseng, V.S., Shie, B.E., Wu, C.W., Philip, S.Y.: Efficient algorithms for mining high utility itemsets from transactional databases. IEEE Transactions on Knowledge and Data Engineering 25(8), 1772–1786 (2012)
37. Tseng, V.S., Wu, C.W., Fournier-Viger, P., Philip, S.Y.: Efficient algorithms for mining top-k high utility itemsets. IEEE Transactions on Knowledge and Data Engineering 28(1), 54–67 (2015)
38. Tseng, V.S., Wu, C.W., Shie, B.E., Yu, P.S.: Up-growth: an efficient algorithm for high utility itemset mining. In: Proceedings of the 16th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. pp. 253–262 (2010)
39. Wang, K., Wu, J.M.T., Cui, B., Lin, J.C.W.: Revealing top-k dominant individuals in incomplete data based on spark environment. In: International Conference on Genetic and Evolutionary Computing. pp. 471–480. Springer (2021)
40. Wu, J.M.T., Lin, J.C.W., Tamrakar, A.: High-utility itemset mining with effective pruning strategies. ACM Transactions on Knowledge Discovery from Data (TKDD) 13(6), 1–22 (2019)
41. Wu, J.M.T., Liu, S., Lin, J.C.W.: Efficient uncertain sequence pattern mining based on hadoop platform. Journal of Circuits, Systems and Computers (2022)
42. Wu, J.M.T., Teng, Q., Srivastava, G., Pirouz, M., Lin, J.C.W.: The efficient mining of skyline patterns from a volunteer computing network. ACM Transactions on Internet Technology (TOIT) 21(4), 1–20 (2021)
43. Wu, J.M.T., Zhan, J., Lin, J.C.W.: An aco-based approach to mine high-utility itemsets. Knowledge-Based Systems 116, 102–113 (2017)
44. Yao, H., Hamilton, H.J.: Mining itemset utilities from transaction databases. Data & Knowledge Engineering 59(3), 603–626 (2006)
45. Yao, H., Hamilton, H.J., Butz, C.J.: A foundational approach to mining itemset utilities from databases. In: Proceedings of the 2004 SIAM International Conference on Data Mining. pp. 482–486. SIAM (2004)
46. Yeh, J.S., Li, Y.C., Chang, C.C.: Two-phase algorithms for a novel utility-frequent mining model. In: Pacific-Asia Conference on Knowledge Discovery and Data Mining. pp. 433–444. Springer (2007)
47. Zaki, M.J., Parthasarathy, S., Ogihara, M., Li, W.: Parallel algorithms for discovery of association rules. Data Mining and Knowledge Discovery 1(4), 343–373 (1997)
48. Zaki, M.J.: Scalable algorithms for association mining. IEEE Transactions on Knowledge and Data Engineering 12(3), 372–390 (2000)

49. Zida, S., Fournier-Viger, P., Lin, J.C.W., Wu, C.W., Tseng, V.S.: Efim: a highly efficient algorithm for high-utility itemset mining. In: Mexican International Conference on Artificial Intelligence. pp. 530–546. Springer (2015)

Jimmy Ming-Tai Wu received the Ph.D. degree with major in computer science and engineering from National Sun Yat-sen University, Kaohsiung, Taiwan. He is currently an Associate Professor with the College of Computer Science and Engineering, Shandong University of Science and Technology. He was an Assistant Professor with the School of Computer Science and Technology, Harbin Institute of Technology, Shenzhen, China. He was with an IC design company in Taiwan as a Firmware Developer and Information Technology Manager for two years. He was also a Research Scholar with the Department of Computer Science and Information Engineering, National University of Kaohsiung, Kaohsiung, Taiwan, and with the Department of Computer Science, College of Engineering, University of Nevada, Las Vegas. His current research interests include data mining, big data, cloud computing, artificial intelligence, evolutionary computation, machine learning, and deep learning.

Ranran Li is currently pursuing the M.S. degree in Shandong University of science and technology, Qingdao, China. Her current research interests include data mining and big data.

Pi-Chung Hsu received his doctor degree in information engineering at national Sun Yat-sen University Taiwan in 2003. Now he is the associate professor at Shu-Te University Taiwan. His research directions include computer graphics, implicit surfaces and solid modeling. Dr. HSU may be reached at pichung@stu.edu.tw

Mu-En Wu is an Associate Professor at Department of Information and Finance Management at National Taipei University of Technology, Taiwan. Dr. Wu received his Ph.D. degree with major in computer science from National Tsing Hua University, Taiwan, in 2009. After that, he joined Institute of Information Science, Academia Sinica at Taipei City, Taiwan as a postdoctoral fellow during 2009 2014. During February 2014 to July 2017, he served as an assistant professor of Department of Mathematics at Soochow University. He has a wide variety of research interests covering cryptography, information theory, prediction market, money management, and financial data analysis. He has published more than 100 research papers in referred journals and international conferences

Received: June 15, 2022; Accepted: March 27, 2023.