The Efficient Mining of Skyline Patterns from a Volunteer Computing Network

JIMMY MING-TAI WU, Shandong University of Science and Technology, China
QIAN TENG, Shandong University of Science and Technology, China
GAUTAM SRIVASTAVA, Brandon University, Canada and China Medical University, Taiwan
MATIN PIROUZ, California State University, U.S.A
JERRY CHUN-WEI LIN*, Western Norway University of Applied Sciences, Norway

In the ever growing world, the concepts of High-utility Itemset Mining (HUIM) as well as Frequent Itemset Mining (FIM) are fundamental works in knowledge discovery. Several algorithms have been designed successfully. However, these algorithms only used one factor to estimate an itemset. In the past, skyline pattern mining by considering both aspects of frequency and utility has been extensively discussed. In most cases, however, people tend to focus on purchase quantities of itemsets rather than frequencies. In this paper, we propose a new knowledge called skyline quantity-utility pattern (SQUP) to provide better estimations in the decision-making process by considering quantity and utility together. Two algorithms respectively called SQU-Miner and SKYQUP are presented to efficiently mine the set of SQUPs. Moreover, the usage of volunteer computing is proposed to show the potential in real supermarket applications. Two new efficient utility-max structures are also mentioned for the reduction of the candidate itemsets respectively utilized in SQU-Miner and SKYQUP. These two new utility-max structures are used to store the upper-bound of utility for itemsets under the quantity constraint instead of frequency constraint, and the second proposed utility-max structure moreover applies a recursive updated process to further obtain strict upper-bound of utility. Our in-depth experimental results prove that SKYQUP has stronger performance when a comparison is made to SQU-Miner in terms of memory usage, runtime and the number of candidates.

CCS Concepts: • **Information systems → Association rules**; *Data analytics*; • **Computing methodologies → Knowledge representation and reasoning**.

Additional Key Words and Phrases: Skyline quantity-utility patterns (SQUPs); data mining; utility-quantity list; utility-max

ACM Reference Format:
Jimmy Ming-Tai Wu, Qian Teng, Gautam Srivastava, Matin Pirouz, and Jerry Chun-Wei Lin*. 2020. The Efficient Mining of Skyline Patterns from a Volunteer Computing Network. 1, 1 (September 2020), 20 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

*Corresponding author.
Authors' addresses: Jimmy Ming-Tai Wu, Shandong University of Science and Technology, 579 Qianwangang Rd, Qingdao, Shandong, 266590, China, wmt@wmt35.idv.tw; Qian Teng, Shandong University of Science and Technology, 579 Qianwangang Rd, Qingdao, Shandong, 266590, China, qrape@foxmail.com; Gautam Srivastava, Brandon University, 270 18th St, Brandon, MB R7A 6A9, Canada, Research Centre for Interneural Computing, China Medical University, Taichung, Taiwan, srivastavag@brandonu.ca; Matin Pirouz, California State University, Davis Street, Fresno, CA, U.S.A, mpirouz@ieee.org; Jerry Chun-Wei Lin*, Western Norway University of Applied Sciences, Inndalsveien 28, Bergen, 5063, Norway, jerrylin@ieee.org.

XXXX-XXXX/2020/9-ART $15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Knowledge discovery can be defined as the process to obtain knowledge from information of various kinds with accordance with user needs that may also vary. The main purpose of what is defined as "knowledge discovery" is protecting users with mundane details of raw data and be able to extract novel, effective, pertinent knowledge from that original (raw) data. FIM, short for frequent itemset mining, is a very important research foundation in knowledge discovery [1, 2, 28, 41]. It can tell us the variables that appear together frequently within the dataset, as well as provide some support for possible decision-making. So the aim of FIM is to discover itemsets that occur together and appear in at least minimum supplemental transactions of the database, where minimal supplemental (minsup) is a threshold that defined by users. So far, FIM has been widely used, such as shopping basket data analysis, web page prefetching, cross shopping, personalized website, network intrusion detection, etc.

However, FIM is known to only consider the frequency of the items while assuming that all known items occur to a maximum value of once within a given transaction. However, FIM is only able to reflect whether a given item occurs within a transaction. Moreover, any interesting factors such as unit profit as well as weight of items are not being considered. But in practical applications, for example, retailers often focus on benefits. Thus, diamond ring sales may be less frequent than department store clothing, but the former unit sales profit is much higher. As a result, the frequency of an item/set is not sufficient to identify high utility patterns. In order to overcome this limitation, high-utility itemsets mining (HUIM) [6, 25, 35, 39] has become the focus of data mining. It was aimed at discovering itemsets that have a high utility much higher than a user-defined minimum utility threshold. Chan *et al.* [6] first discovered utility mining problem and then Yao *et al.* [39] followed with interest of the quantity of items and the unit profit of items to discover the set of HUIs. Disappointingly, HUIM yield a more difficult problem than FIM because the utility measure is not known as anti-monotonic (i.e., if a pattern is not considered as a satisfied pattern, any of its supersets will not be considered either). Furthermore, utility of a given itemset may actually be greater, smaller, or even equal to the utility of its supersets. To address this limitation, Liu *et al.* [25] proposed a new model to satisfy downward closure property called transaction-weighted utility (TWU), thus being able to reduce search space for mining HUIs. Soon after, Lin *et al.* [20] proposed a tree-based structure to maintain necessary information of the database, that is high-utility pattern (HUP)-tree. Subsequently extensively studied [3, 12, 15, 32, 37] were conducted and several algorithms were proposed, which imply that high-utility items mining (HUIM) has become the focus of data mining.

All the algorithms mentioned above are used to create a large set of rules that cab be used for decision-making, but that depend on user-set parameters. Although, users were concerned with finding more concise rules in order to choose the most appropriate rules to reduce the time to make effective decisions. Top-k association-rule mining [11] and high-utility mining [35] were designed for finding more concise rules. Although the time of making effective decisions was reduced but only concern one aspect such as frequency or utility of items is not meet the demands of users. In real-life situations, considering two or greater number of factors jointly in decision-making seems more than necessary. As an example, if we consider the speed of a given vehicle alongside the cost of the vehicle, if you choose the plane, the price of the plane should be higher than the bus, but the speed of the plane also much faster than the bus. In the process of mining if top-k HUIs or ARs are only given consideration of speed/price of the vehicle, obviously that is not insufficient. To solve the above limitations, Goyal *et al.* [17] first proposed a new algorithm called SKYMINE to mine skyline frequent-utility (SFU) itemsets, which itemsets are non-dominated by any other itemset under the aspects of frequency and utility. Note that this method also has disadvantages,

such as SKYMINE is based on the UP-tree structure [36] , but this approach may generate a huge amounts of candidates. For the sake of improving the performance for obtaining the skyline, Pan *et al.* [26] presented the SFU-Miner, but it still has the limitation in terms of runtime. To address this limitation, Lin *et al.* [21] proposed efficient utility-list-based skyline mining algorithm (named SKYFUP-B/SKYFUP-D) to discover SFUPs, short for skyline frequent-utility patterns. Any of the above algorithms considered frequency of items, but in real-life situations, people tend to focus on quantity of items. For example, in case of stock squeeze, retailers are concerned about the sales of items, not the frequency that items occur in transactions.

In the previous works, they seldom described an applied scenario in a real application using mining algorithms as proposed. Because of the high performance of proposed methods, a volunteer computing network employing in a supermarket scenario is proposed to be applied to the suggested algorithms. Nowadays, a cell phone is the standard accessory for everybody and carries on high performance computing unit and cell network transportation ability. If each shopping cart in a supermarket has the ability to connect to the customers' cell phones, the supermarket can utilize these cell phones collecting the information to reveal useful real-time patterns and provide more convenient services to their customers. The detailed description is provided in the following section. Major contributions are listed below.

(1) A new pattern called skyline quantity-utility pattern (SQUP) is presented in this paper. It is composed of itemsets that are non-dominated by any other itemset under the aspects of quantity and utility. Meanwhile, two efficient algorithms are designed to discover SQUPs called SQU-Miner and SKYQUP.
(2) Utility-max structure was proposed to be able to allow max utility of itemsets under frequency. In this paper, utility-max is implemented to allow max utility of itemsets to remain under quantity constraint as well as being able to reduce the search space so users do not require setting up minimum quantity threshold for discovering SQUPs.
(3) Meanwhile, we expanded utility-list structure and added quantity elements to the list. Thus it forms a new list structure called utility-quantity-list. This list is used to store the necessary information of both quantity and utility in place of UP-tree structure that are used in SKYMINE.

We organize the rest of this work in the following manner. Some related works are mentioned in Section 2. The key preliminaries and the problem statement of skyline quantity-utility patterns Mining (SQUPM) are given in Section 3. In Section 4, we propose SQU-Miner and SKYQUP algorithms and introduce our redesigned utility-quantity-list. An illustrated example were proposed in Section 5. An experimental evaluation of the two mentioned algorithms are provided in Section 6 and some conclusions are obtained by comparing this two methods. We end the paper with some concluding remarks as well as future directions are finally presented in Section 7.

2 LITERATURE SURVEY

In this section, we will introduce related works about high-utility itemset mining(HUIM) and the concept of skyline.

2.1 High-Utility Itemset Mining

In the past few years, pattern-mining algorithms [9, 10, 13, 14, 38] such as FIM has attracted the attention of the data mining industry, as FIM is the first step in association rule mining (ARM). Researchers have put forward a wide range of research methods to mine frequent itemsets, and divided it into three categories: hierarchical growth model [1, 28, 30], pattern growth model [18, 23], and Eclat model [31, 42]. For the first approaches, Apriori algorithm [1] is the mostly popular

algorithm but candidate sets are generated monotonically at each level. In order to improve the mining performance, frequent-pattern (FP)-growth was designed as the fundamental pattern growth methods. In addition, Eclat model uses the vertical tid-list database instead of generic Apriori-like and FP-tree-like approaches to process the database. The depth-first search is then utilized in Eclat for mining the frequent patterns efficiently. However, all of the above algorithms only consider item's frequency and assumes that all item occur at most once in transaction. Thus, FIM is only able to reflect the occurrence frequency of an item/set in a iven transaction; on the other hand, many interesting factors such as weight, unit profit of the items are not considered. But in practical applications, for example, retailers often focus on benefits. Diamond ring sales may be less frequent than bread, but the former unit sales profit is much higher. As a result, the frequency of an item/set is not sufficient to identify high utility patterns.

In order to solve the defects mentioned above, HUIM has become the focus of data mining which considers both quantity and unit profit of items. If the utility of an itemset is not less than the user-defined minimum utility threshold, it is defined as a HUI. Based on the mathematical properties of utility constraints, Yao *et al.* [39] presented an efficient algorithm to mining HUI. Unfortunately, the designed algorithm can not maintain the downward closure (DC) property, so it can not find the complete HUI sets. Later, Liu *et al.* [25] proposed a new concept called transaction weighted utility (TWU) and designed a new set called high transaction-weighted utility itemset, or HTWUIs, which is based on transaction-weighted downward closure, which is also known as the TWDC property. Nevertheless, using TWU to mine HUIs will need multiple database scans and produce amount of search space. Besides, Liu and Qu [24] proposed a novel list-based algorithm to find the HUIs called HUI-Miner. The novel algorithm discovered the set of the potential HUIs use the simple join operation, and effectively reduce the generation of candidate sets and the execution time. In this algorithm, a new list structure was designed to store the necessary information of both frequency and utility called utility-list structure. Relevant contents about utility-list structure will be introduced in detail in the fourth part. Additionally, extensively and various studies [3, 12, 15, 32, 34, 36, 37] were proposed to efficiently discover the HUIs under specific domains and applications.

2.2 The Previous Hybrid Approach

Although FIM and HUIM can efficiently find itemsets with high frequency and high utility, but none of them concern both utility as well as frequency together to mine the effective information. To discover the set of itemsets with both high frequency and high utility, Yeh *et al.* [40] first presented a algorithm called two-phase algorithm. However in this algorithm, it is necessary for users to define two thresholds called minimum support and utility threshold respectively. In order to improve the performance of the algorithm, Podpecan *et al.* [29] proposed a faster algorithm to discover itemsets with high frequency and high utility, but it also need to set two thresholds. Later, Goyal *et al.* [17] proposed a new pattern and named it skyline frequent-utility pattern (SFUP), in which each itemset is non-dominated by others by concerning both frequency and utility contains. A highly efficient algorithm known as SKYMINE was presented to mine the required SFUPs. Regretfully this approach is based on UP-tree structure and generate amount of candidates, so it is not an efficient algorithm for reducing the search space. To address this limitation, Lin *et al.* [21] proposed two algorithms to discover skyline frequent-utility pattern based on the well known utility-list structure named SFU-Miner and SKYFUP-D respectively.

2.3 The Skyline Concept

When in itself, skyline is known to represent a set of points, each of which is based on a multidimensional non dominating point. It is very important to deal with large-scale database

because it only returns non-dominating points as the solution of decision. For example, suppose clearly that when observed as a known set of objects, we can say that the skyline of a given object may refer as well to objects that may not be governed using any other given object. If and only if at least one object can dominate another known object, this fact alone indicates clearly that the object is as good or even better weighting all dimensions and is at least better in one dimension than all other objects. In real-world applications, we can say that the distance between a given hotel and its own city center is in clear contrast to what the price of rooms in the hotel is. In other words, if and only if the given hotel is very close to its city center, reservation price of rooms in that hotel is greater than a hotel far away from its city center. We show an example of this fact in Fig. 1. In Fig. 1, it can be seen that a given set of hotels are used for booking. The X-axis is used to clearly show the location of a hotel when compared to city center as well as price of hotel rooms being expressed on the horizontal Y-axis. Through this example, we are able to find that skyline points are g, b as well as m respectively, as they are not dominated by others under distance and price contains.

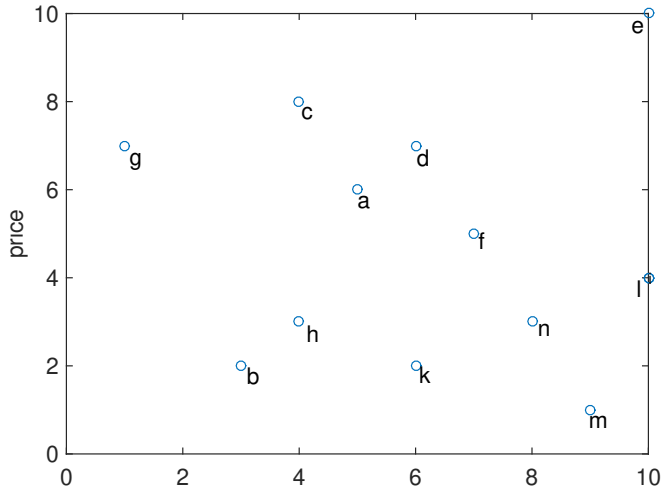


Fig. 1. A skyline example.

The concept of skyline was firstly proposed using a method of "divide and conquer" to identify point sets. Borzsony *et al.* [4] the presented skyline operation in the context of databases as well as gave an evaluation of skyline points using B-tree as well as R-tree. An improved block nesting loops is proposed by Chomicki *et al.* [7] to improve the performance by using a certain tuple order. Tan *et al.* [33] implemented a new well-known algorithm that is able to output skyline points step-by-step without the need to scan the entire sets of input data. Kossmann *et al.* [19] designed a NN (neural network) algorithm that is based on k-nearest neighbor searche as well as applying again "divide and conquer" and R-tree technology to find all of the available skyline points. Papadias *et al.* [27] created a "branch and bound" skyline algorithm, known as BBS, that is based on k-nearest neighbor search that is able to clearly perform single access used for discovering skyline points. Other related works have been extensively studied and discussed [5, 22].

In the previous works, we see that research and academics have seldom described any applied scenario pertinent to the real-world applications. Because of the high performance of proposed methods, a volunteer computing network employing in a supermarket scenario is proposed to be applied to the suggested algorithms. Nowadays, a cell phone is the standard accessory for everybody and carries on high performance computing unit and cell network transportation ability.

If each shopping cart in a supermarket has the ability to connect to the customers' cell phones, the supermarket can utilize these cell phones collecting the information to reveal useful real-time patterns and provide more convenient services to their customers. This scenario can be applied into any pattern-mining tasks to reveal more useful information for decision-making. Moreover, in the traditional pattern-mining tasks such as FIM or HUIM, they all consider one aspect of mining information, obviously, it is not enough. However, skyline method can find the non-dominated solution based on multi-dimensions. In this paper, a new pattern called skyline quantity-utility pattern (SQUP) is proposed and designed to provide better solutions in the decision-making process by considering both quantity and utility together. To discover SQUP, two efficient algorithms were designed called SQU-Miner and SKYQUP respectively. Besides, we use the utility-max structure to keep the max utility of the items under the frequency and use it to reduce search space. Meanwhile, a new structure called utility-quantity-list is designed and used to store the essential information of both quantity and utility in the developed model.

3 PRELIMINARY AND PROBLEM STATEMENT

Table 1. A quantitative database.

T_{ID}	**Transactions (Item/quantity)**
T_1	$B/1,C/2,D/1$
T_2	$A/4,B/1,C/3,D/1,E/1$
T_3	$A/4,C/2,D/1$
T_4	$C/2,D/1,E/1$
T_5	$A/5,B/2,D/1,E/2$
T_6	$A/3,B/4,C/1,D/1$
T_7	$D/1,E/1$

Table 2. Profits.

Items	C	B	A	D	E
Profit	1	2	1	5	4

Table 3. A sorted quantitative database.

T_{ID}	**Transaction (Item/quantity)**
T_1	$B/1,C/2,D/1$
T_2	$E/1,B/1,C/3,A/4,D/1$
T_3	$C/2,A/4,D/1$
T_4	$E/1,C/2,D/1$
T_5	$E/2,B/2,A/5,D/1$
T_6	$B/4,C/1,A/3,D/1$
T_7	$E/1,D/1$

3.1 Preliminaries

Let $I = \{i_1, i_2, \ldots, i_m\}$ be defined as a set containing a total of distinct items m. Quantitative database $D = \{T_1, T_2, \ldots, T_n\}$ is defined as a transaction set, where each and every transaction is also a subset of I containing a unique identifier t, called T_{ID}. Furthermore, each item i_j within a transaction T_q contains its purchase quantity (internal utility) and denoted as $q(i_j, T_q)$. A profit table stores the profit value of each item i_j. k-itemset is a set of k distinct items $X = \{i_1, i_2, \ldots, i_k\}$, and k is the length of the itemsets. If $X \subseteq T_q$ that is to say an itemset X is contained in a transaction T_q. In this paper, we use Table 1 as an example to evaluate our developed algorithms. From Table 1, it contains seven transactions as well as give distinct items respectively shown as (A) to (E) and profit values through external utility are then clearly shown in Table 2 through a profit table.

Definition 3.1. In a database D, in transaction T_q, the purchase of an itemset X is denoted as $q(X, T_q)$ and defined as:

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

Through Table 1, in transaction T_1, the quantity of the item (B) is 1 and the quantity of the item (C) is 2, so the quantity of the itemset (BC) is calculated as 1.

Definition 3.2. Define a transaction T_q, the utility of an item i_j is denoted as $u(i_j, T_q)$ and defined as:

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

Through Table 1, in transaction T_1, the utility of the item (B) can be calculated as $u(B, T_1) = q(B, T_1) \times pr(B) = 1 \times 2 = 2$, the utility of the item (D) can be calculated as $u(D, T_1) = q(D, T_1) \times pr(D) = 1 \times 5 = 5$. In transaction T_2, the utility of the item (A) can be calculated as $u(A, T_2) = q(A, T_2) \times pr(A) = 4 \times 1 = 4$.

Definition 3.3. Define a transaction T_q, the utility of an itemset (X) is denoted as $u(X, T_q)$ and defined as:

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

Through Table 1, in transaction T_1, the utility of the itemset (D) is calculated as $u(D, T_1) = q(D, T_1) \times pr(D) = 1 \times 5 = 5$, the utility of the itemset (BCD) is calculated as $u(BCD, T_1) = q(B, T_1) \times pr(B) + q(C, T_1) \times pr(C) + q(D, T_1) \times pr(D) = 1 \times 2 + 2 \times 1 + 1 \times 5 = 9$.

Definition 3.4. Define in a database D, the utility of itemset X can be denoted as $u(X)$ and defined as:

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

Through Table 1, the itemset (D) appears in transactions $T_1, \ldots, T_7$, so the utility of the itemset (D) is calculated as $u(D) = u(D, T_1) + u(D, T_2) + \ldots + u(D, T_7) = 5 + 5 + 5 + 5 + 5 + 5 + 5 = 35$. The itemset (BCD) appears in transactions T_1, T_2, T_6, so the utility of the itemset (BCD) is calculated as $u(BCD) = u(BCD, T_1) + u(BCD, T_2) + u(BCD, T_6) = 9 + 10 + 14 = 33$.

Definition 3.5. In a database D, the utility of a transaction is denoted as $tu(T_q)$ and defined as:

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

As another example using Table 1, transaction T_1 contains item (C), (B) and (D), so $tu(T_1) = u(B,T_1)+u(C,T_1)+u(D,T_1) = 2+2+5 = 9$. Similarly transactions in a resting phase from $T_2,\ldots,T_7$ can be calculated as $tu(T_2) = 18$, $tu(T_3) = 11$, $tu(T_4) = 11$, $tu(T_5) = 22$, $tu(T_6) = 17$, and $tu(T_7) = 9$.

As we mentioned above, HUIM does not does not hold the downward closure property, Liu *et al.* [25] was able to give the transaction-weighted utility twu model which holds the transaction-weighted-utilization downward closure (TWUDC) property. In this model, each itemset X has utility within a transaction can be enlarged into tu of the transaction, so the twu is a upper-bound. The definition of twu is given below.

Definition 3.6. In database D, the transaction-weighted utility of an itemset X is denoted as $twu(X)$ and defined as:

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

Shown again through example created using Table 1, an itemset shown as (D) can be seen to appear in transactions $T_1,\ldots,T_7$. Therefore, the twu of itemset (D) can calculated as $twu(D) = tu(T_1)+\ldots+tu(T_7) = 9+18+11+11+22+17+9 = 97$.

To discover skyline patterns by considering both the quantity and utility factors together, we define the skyline quantity-utility pattern mining (SQUPM) as follows.

Definition 3.7. For two itemsets X and Y, if $q(X) \geq q(Y)$ and $u(X) > u(Y)$ or $q(X) > q(Y)$ and $u(X) \geq u(Y)$, we can say an itemset X dominates an itemset Y which is denoted as $X \succ Y$.

Definition 3.8. If an itemset X cannot be shown to be dominated by another itemset from within the database through consideration of the quantity as well as utility factors, we can say that X is a skyline quantity-utility pattern (SQUP).

3.2 Problem Statement

When we look at the above mentioned definitions, we can clearly define our problem of SQUPM as the ability to find skyline patterns which are both required to be under quantity as well as utility restrictions.

For our given example as in Table 1, the quantity as well as utility of (D) are calculated as 7 and 35, respectively. For (A), they are calculated as 16 and 16 respectively. The quantity/utility of (ED) are calculated as 4 and 40 respectively. Finally the quantity/utility of (BAD) are calculated as 3 and 41 respectively. The (D), (A), (ED) and (BAD) may as well be taking into consideration as SQUPs due to the fact that none of these mentioned itemsets can be shown to be dominated (non-dominated) with any of the other itemsets that may exist within a given database.

4 SKYLINE QUANTITY-UTILITY PATTERN MINING

There are two independent parts provided in this section. The first part is the volunteer computing network scenario using the proposed skyline framework in a supermarket. It provides a flowchart of the applied scenario and the detailed descriptions in each part. The second part focuses on the skyline mining algorithms to reveal the non-dominated patterns in a dataset. The detailed pseudo-code of the developed algorithms are shown in the following subsection.

4.1 Real-time skyline patterns framework in supermarkets

An introduced scenario of volunteer computing network in a supermarket is shown in Fig. 2. Recently, cell phones are the standard accessories of everyone and provide high-performance computing resources. Moreover, due to the widespread cell phone network service, cell phones are

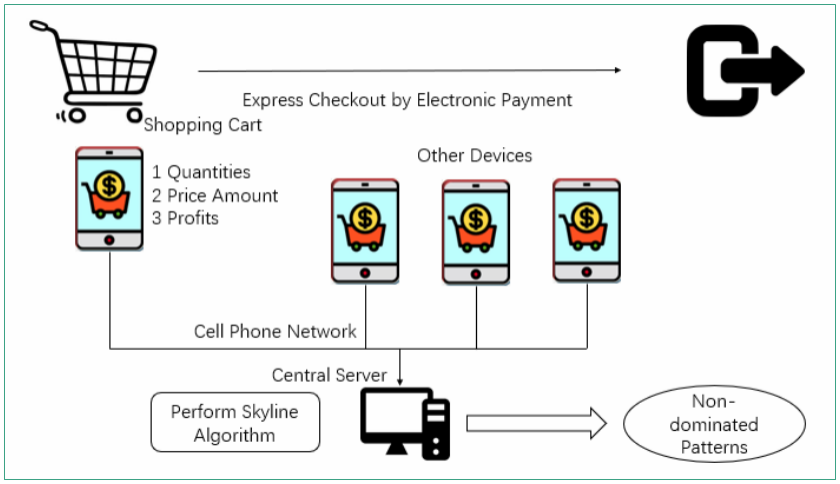
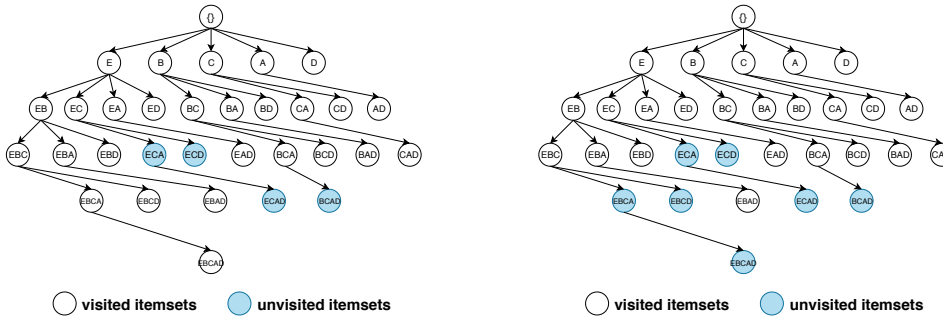


Fig. 2. Real-time skyline patterns framework in supermarkets.

very suitable to be the devices of Volunteer Computing resources. In this scenario, a supermarket utilizes a volunteer computing network composed of customers' cell phones to provide better services to customers. A novel designed shopping cart in this framework equipped Bluetooth[1] connection ability and NFC[2] technique. The produces in this supermarket all attract NFC tags and shopping carts can summarize the quantities and list of the purchased produces in a shopping cart. The connected cell phone is responsible for calculating the total quantities of products, and the total purchased price and the total profits are in this shopping cart. A central server collects the information from all cell phones by cell phone network service. Finally, the central server can perform the proposed skyline algorithms and obtain real-time non-dominated patterns. Due to the credit of sharing computing resources, the customers can attain express checkout service and the supermarket also can provide an extra discount to encourage customers to share their devices and using this service.

4.2 Skyline Algorithms

When analyzing the skyline algorithms as they are proposed here and as shown in Fig. 3, we use a tree to represent the search space to be used in the mining for SQUPs and use the common technique of "depth-first search" for exploration of a given search space for the sake of mentioned itemsets on the utility-quantity-list (UQL) are sorted in *twu*-ascending order. The utility-quantity-list (UQL) architecture is made to be able to store a collection of essential information which includes both quantity as well as utility. The definition of the UQL structure will be discussed next. Besides the structure (utility-max) can be used to maintain max utility for itemsets below quantity constraint and reduce further the size of the search space, next two different utility-max structures will be discussed and later two algorithms will be proposed to discover skyline quantity-utility patterns (SQUPs) based on the two different utility-max structures.



(a) SQU-Miner (b) SKYQUP

Fig. 3. The searching tree of the proposed methods.

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

(a) E

t_{id}	quan	iutil	rutil
2	1	4	14
4	1	4	7
5	2	8	14
7	1	4	5

(b) B

t_{id}	quan	iutil	rutil
1	1	2	7
2	1	2	12
5	2	4	10
6	4	8	9

(c) C

t_{id}	quan	iutil	rutil
1	2	2	5
2	3	3	9
3	2	2	9
4	2	2	5
6	1	1	8

(d) A

t_{id}	quan	iutil	rutil
2	4	4	5
3	4	4	5
5	5	5	5
6	3	3	5

(e) D

t_{id}	quan	iutil	rutil
1	1	5	0
2	1	5	0
3	1	5	0
4	1	5	0
5	1	5	0
6	1	5	0
7	1	5	0

4.3 Utility-Quantity-List Structure

Taking database D and sorting its contents using an ascending order for TWU, and represented by $\triangleright$. Each itemset X has its own utility-quantity-list (UQL) structure keeping a certain number of tuples where we can say that each and every tuple has four elements given as $(tid, quan, iutil, rutil)$. Here we define tid as transaction ID containing itemset X, $quan$ as purchase quantity of itemset $X \in tid$, the $iutil$ as utility of itemset $X \in$ transaction tid and finally $rutil$ as the sum of the utilities of all itemsets after X in transaction tid and defined as:

Definition 4.1. $rutil(X) = \sum_{i_j \subseteq T_q / X} u(i_j, T_q)$

[1]https://en.wikipedia.org/wiki/Bluetooth
[2]Near-field communication, https://en.wikipedia.org/wiki/Near-field_communication

Because the procedure to construct utility-quantity-list structure is the same as HUI-Miner, readers can refer to [24]. Again focusing on Table 1, we see that the *twu*-ascending order for items can be seen as $E \rhd B \rhd C \rhd A \rhd D$. According to HUI-Miner algorithm, all 1-itemsets utility-quantity-list structures are constructed. The UQL structure of itemset (E) is $UQL.E = \{(T_2, 1, 4, 14), (T_4, 1, 4, 7), (T_5, 2, 8, 14), (T_7, 1, 4, 5)\}$.

4.4 Utility-Max Structures

In this part, two efficient utility-max structures are mentioned to record the max utility of the potential itemsets, thus reducing the search space for finding the SQUPs from the candidates.

Definition 4.2. The first utility-max structure stores the max utility for each quantity i is denoted $Umax1(i)$ and defined as:

$$Umax1(i) = max\{u(X) \mid q(X) = i\} \tag{7}$$

Definition 4.3. The second utility-max structure stores the max utility of the patterns if their quantity is greater than or can also be equal to index parameter i as:

$$Umax2(i) = max\{u(X) \mid q(X) \geq i\} \tag{8}$$

These two utility-max structures are used to store the max utility of itemsets under any quantity constraint and reduce search space while mining SQUPs. The first *Umax1* is used in the developed SQU-Miner and the second *Umax2* is used in the designed SKYQUP. To reduce the redundant and overlapping contents of the algorithm, *Umax1* and *Umax2* are then simplified as the *Umax* for the later contents. If the SQU-Miner is chosen to mine the required SQUPs, then the *Umax* in Algorithm 1 is considered as the *Umax1*, and if the SKYQUP is performed for mining the SQUPs, the *Umax* in Algorithm 1 is then utilized by the *Umax2*. Experiments will be later performed to show that the SKYQUP with *Umax2* is more effective to reduce the search space for finding SQUPs.

4.5 Pruning Strategy

Theorem 4.4. *For a given itemset X, we say that if and only if sum of iutil within utility-quantity-list of $X < Umax(q(X))$, $X \notin SQUP$.*

Theorem 4.5. *For a given itemset X, we say that if the sum of iutil and rutil in the utility-quantity-list of $X < Umax(q(X))$, then say that all the superset of X are not able to be SQUPs.*

Again, note that *Umax1* or *Umax2* is then simplified as *Umax* in two theorems depending on which algorithm (SQU-Miner or SKYQUP) is used for mining SQUPs. Using these 2 lemmas, we define the following strategy for pruning the search space as:

Two pruning strategies:

(1) If the sum of *iutil* in the utility-quantity-list of X is less than the $Umax(q(X))$, we can clearly cut X from search space.

(2) If the sum of *iutil* and *rutil* in the utility-quantity-list of X is less than the $Umax(q(X))$, we can cut all supersets of X off the search space.

4.6 The Mining Algorithms

In this part, based on the two utility-max structures mentioned above, we design two algorithms named SQU-Miner (based on the *Umax1*) and SKYQUP (based on the *Umax2*) respectively to discover skyline utility-quantity patterns (SQUPs). Meanwhile, these two algorithms are all based on depth-first search, because all items are sorted by *twu* in ascending order. The only difference between the two algorithms is the process of judgment given in Algorithm 3 as well as Algorithm 4, respectively. The detailed description of the designed algorithms are as follows:

Algorithm 1 SQU-Miner/SKYQUP algorithm

Input:
D, a transaction database; and *ptable*, a profit table.
Output:
The set of skyline quantity-utility patterns (SQUPs).

```
1: ∀i_j ∈ T_q ∧ T_q ∈ D do
2: for each transaction T_q ∈ D do
3:    calculate tu(T_q);
4:    for each item i_j ∈ T_q do
5:       calculate twu(i_j);
6:    end for
7: end for
8: sort i_j in twu-ascending order;
9: re-organize the database D;
10: UQLs ← construct(i_j);
11: set k is the maximum quantity of i_j in D;
12: for i = 1 to k do
13:    initialize Umax(i) to 0; {SQU-Miner uses Umax1 and SKYQUP uses Umax2, refer to Section
       4.4}
14: end for
15: SQUPs ← null;
16: Search (null, UQLs, Umax, SQUPs);
17: return SQUPs;
```

In Algorithm 1, the first step is shown to be the calculation of transaction-weighted utility for each item followed by sorting each item by *twu*-ascending order, later re-organize the database D according to the *twu-ascending* order of each item. After that, the utility-quantity-list of each item is constructed based on the HUI-Miner algorithm. Next, initialize the *Umax* (*Umax1* and *Umax2*) values of varied quantity from 1 to the *max* quantity as 0. After updating SQUPs, the *Umax* (*Umax1* and *Umax2*) is updated at the same time. But note that for two different *Umax* structures, the update operations of *Umax* (*Umax1* and *Umax2*) are also different. Last, it is necessary to check whether an itemset is still a SQUP after updating SQUPs. The details of **Judge 1** and **Judge 2** are shown in Algorithm 3 as well as Algorithm 4 respectively. The purpose of this step is to reduce the search space for mining SQUPs. Then the search algorithm, shown in Algorithm 2, is designed to explore the search space, and details are given below.

From Algorithm 2, we see the process for "depth-search" that can be used for mining SQUPs. For each itemset $X \in UQLs$, if the $\sum$ of *iutil* $\in$ the utility-quantity-list of X not less than $Umax(q(X))$, then X may be a SQUP according to Lemma 1, so X needs to be further determined whether it is a SQUP. The **Judge** function as shown in Algorithm 3 is detailed as below. On the other hand, it is

Algorithm 2 Search

Input:
P.UQL, the utility of an itemset P; *UQLs*, the set of utility-quantity-list of P's all 1-extensions; $Umax(k)$, the max utility of k; *SQUPs*, the set of SQUPs.

```
1:  for each itemset X ∈ UQLs do
2:    if sum(X.iutil) ≥ Umax(q(X)) then
3:      Judge (X, Umax, SQUPs);
4:    end if
5:    if sum(X.iutil) + sum(X.rutil) ≥ Umax(q(X)) then
6:      for each Y ◁ X do
7:        exUQLs ← construct(P.UQL, X, Y);
8:      end for
9:      Search(X, exUQLs, Umax, SQUPs)
10:   end if
11: end for
```

necessary to determine whether the supersets of X needs to be searched. If the $\sum$ of *iutil* and *rutil* $\in$ the utility-quantity-list of X is not less than the $Umax(q(X))$, then the supersets of X needs to be explored based on Lemma 2. Later the set of *UQLs* of all supersets of X is constructed.

Algorithm 3 Judge1

Input:
X, the potential SQUP; $Umax(k)$, the max utility of k; *SQUPs*, the set of SQUPs.

```
1:  ∃Y in SQUPs, q(Y) > q(X)
2:  if Y = null or u(X) > u(Y) then
3:    insert X to SQUPs;
4:    for each n in q(X) do
5:      if Umax(n) < u(X) then
6:        Umax(n) := u(X);
7:      end if
8:      n := n − 1;
9:    end for
10:   for each Y in SQUPs do
11:     if q(Y) < q(X) and u(Y) ≤ u(X) then
12:       remove Y;
13:     end if
14:     if q(Y) = q(X) and u(Y) < u(X) then
15:       remove Y;
16:     end if
17:   end for
18: end if
```

We know from Algorithm 2 that if the $\sum$ of *iutil* in the utility-quantity-list of X is not less than the $Umax(q(X))$, then X may be a SQUP according to Lemma 1, so it is necessary to determine whether X is a SQUP. The first step is to find the first itemset Y which has higher quantity than X in SQUPs, that is to say $q(X) < q(Y)$. Because the sum of *iutil* in the utility-quantity-list of X is not less than the $Umax(q(X))$, so $u(X)$ must higher than or equal to $u(Y)$. Respectively if $u(X) = u(Y)$,

Algorithm 4 Judge2

Input:
X, the potential SQUP; $Umax(k)$, the max utility of k; *SQUPs*, the set of SQUPs.

```
1: ∃Y in SQUPs, q(Y) > q(X)
2: if Y = null or u(X) > u(Y) then
3:    insert X to SQUPs;
4:    Umax(q(X)) := u(X);
5:    for each Y in SQUPs do
6:       if q(Y) < q(X) and u(Y) ≤ u(X) then
7:          remove Y;
8:       end if
9:       if q(Y) = q(X) and u(Y) < u(X) then
10:         remove Y;
11:      end if
12:   end for
13: end if
```

X must not be a SQUP since Y dominates X and if Y is null or $u(X) > u(Y)$, X must be a SQUP. It needs to judge whether an itemset is still a SQUP if its quantity is less than or equal to X.

5 ILLUSTRATIVE EXAMPLE

An illustrative example is presented in this section to only show the procedure of the designed SKYQUP algorithm since SQU-Miner has the similar progress but only the used *Umax* structure is different, we then skip to give the overlapping descriptions for SQU-Miner. As an example again, Table 1 gives a transactional database alongside Table 2 which shows the unit profit of each item respectively. First, *twu* for all items is obtained through calculation using a database scan and the result values are {*twu*(*N*): (*A*):68, (*B*):66, (*C*):66, (*D*):97, (*E*):60}. Second, the collection of all items within the database go through a sorting process in *twu*-ascending order meanwhile the database is re-organized and the results are shown in Table 3. All items' utility-quantity-list structures are constructed in the third step and the results are shown in Table 4. Next, from the *quantity* values in utility-quantity-list of all items, we can get the *maximum* quantity values, so we can initialize *Umax* of each quantity to 0. Next based on the *twu* values of items we can get the searching tree such as Fig. 3 and based on depth-first search, we start to explore search space for mining the required SQUPs.

Let us start with point (E), the quantity of (E) is calculated as 5 and the utility of (E) is calculate as 20 from the utility-quantity-list structure of (E), because $u(E) = 20 > Umax(5) = 0$, so (E) may be a SQUP according to Lemma 1 and insert (E) into SQUPs. Next, the *Umax* values from 1 to 5 are updated to 20, now the *Umax* values of each quantity are $Umax(1) = (2) = (3) = (4) = (5) = 20$. Since we use depth-first search, so the supersets of (E) are considered next. The sum of *rutil*+*iutil* for (E) can be calculated to be 60, clearly higher than $Umax(5) = 20$, so the supersets of (E) are explored using to Lemma 2. Supersets for (E) are (EB), (EC), (EA), (ED), let us first explore itemset (EB). The quantity and utility of (EB) are calculated as 3 and 18 respectively from Table 4. Since $u(EB) = 18 < Umax(3) = 20$ so (EB) must not be a SQUP and the sum of *rutil*+ *iutil* for (EB) can be calculated as 40, clearly higher than $Umax(3) = 20$, so the supersets of (EB) are thus explored. Then the itemset (EBC) is determined. From Table 4 we can get $q(EBC) = 1$ and $u(EBC) = 9$, obviously $u(EBC) = 9 < Umax(1) = 20$, so (EBC) must not be a SQUP. The $\sum$ of *iutil*+*rutil* for (EBC) can be calculated as 18, clearly smaller than $Umax(1) = 20$, therefore it becomes unnecessary

for the exploration of supersets of (EBC) using Lemma 2. Exploration of other itemsets are made continuously until all itemsets are handled. We then discover four skyline quantity-utility patterns (SQUPs), which are (BAD), (ED), (D), (A).

6 EXPERIMENTAL EVALUATION

Table 5. Parameters.

$\|D\|$	Total # of transaction
$\|I\|$	Total # of item
Avglen	Avg length (trans)
MaxLen	Max length (trans)
Type	Set as “Sparse (sp)" or “dense (d)"

Table 6. Characteristics.

ID	Database	$\|D\|$	$\|I\|$	Avglen	MaxLen	Type
1	Mushroom	8,124	119	23	23	d
2	Foodmart	21,557	1550	4	11	sp
3	Retail	88,162	16,470	10	76	sp
4	Chess	3,196	76	37	37	d
5	Accident	4,164	468	25	25	d

Because in this paper, skyline quantity-utility pattern mining is proposed for the first time, so it is not compared with any state-of-the-art methodologies. However, instead in the paper, we proposed two utility-max structures to keep the maximum utility of each quantity and respectively designed two algorithms, so we compared these two approaches in terms of memory usage, runtime, search space size and the number of candidates. It is unreasonable to compare the designed algorithm with the traditional Top-k method, because these methods can only concern one aspect to find the valuable information, and the information found can not get multiple factors to reveal a better solution. The algorithms were implemented in Java and experiments were tested on a computer equipped with an Intel(R) Core(TM) i5-8500 CPU and 8 GB of RAM, running on the 64-bit Microsoft Windows 10 operating system. The proposed algorithms take the following five databases as experimental data called chess [16], mushroom [16], foodmart [8], retail [16] and accident [16]. All databases are stored in plain text files. A row in the file indicate a transaction record, the format of the records is $\{item_1, item_2, \ldots, item_n : \text{total utility} : utility_1, utility_2, \ldots, utility_n\}$ where n is the number of items. A unit utility information for all items is put in a separate file. It can be used to calculate the quantity of each item. Next, we will compare the experimental results in detail. Parameters and characteristics of these datasets are respectively shown in Tables 5 and 6.

6.1 Runtime

Based on two utility-max structure, two algorithms were proposed and were compared with each other on five real-world datasets. In this subsection, the running time for discovering the SQUPs is explored. We give detailed accounting of our experiments in Table 7 and Fig. 4(a) respectively.

Experimental results are given in From Table 7. For the dense (type = d) databases, the proposed SKYQUP algorithms is faster than the proposed SQU-Miner algorithm. For the databases such as

Table 7. Runtime.

Database \ Algorithm	SQU-Miner	SKYQUP
Mushroom	34.59	14.25
Foodmart	1.97	2.05
Retail	610.98	621.76
Chess	433.25	211.28
Accident	1400.87	18.89

chess and mushroom, the proposed SKYQUP algorithm is 2× faster than the proposed SQU-Miner algorithm. But note that for sparse databases such as retail and foodmart, the proposed SQU-Miner algorithm is faster than the SKYQUP algorithm. However, from Fig. 4(a), even though the first method is faster, the running time of the two methods is almost the same. The reasons are as follows. The number # of transactions in retail and foodmart databases is larger. In the second method, it is necessary to update utility-max structure recurrently, so if the database is such large that it needs more time to update utility-max structure. Then, the conclusions are drew as following: If the database is too large the second method is not applicable.

6.2 Memory Usage

Table 8. Memory used.

Database \ Algorithm	SQU-Miner	SKYQUP
Mushroom	64.53	82.73
Foodmart	45.00	67.90
Retail	274.99	244.75
Chess	433.25	211.28
Accident	207.82	57.55

Memory usage of two proposed algorithms using two utility-max structures were compared with each other. Java API is used to measure memory usage and the results are shown in Table 8 and Fig. 4(b).

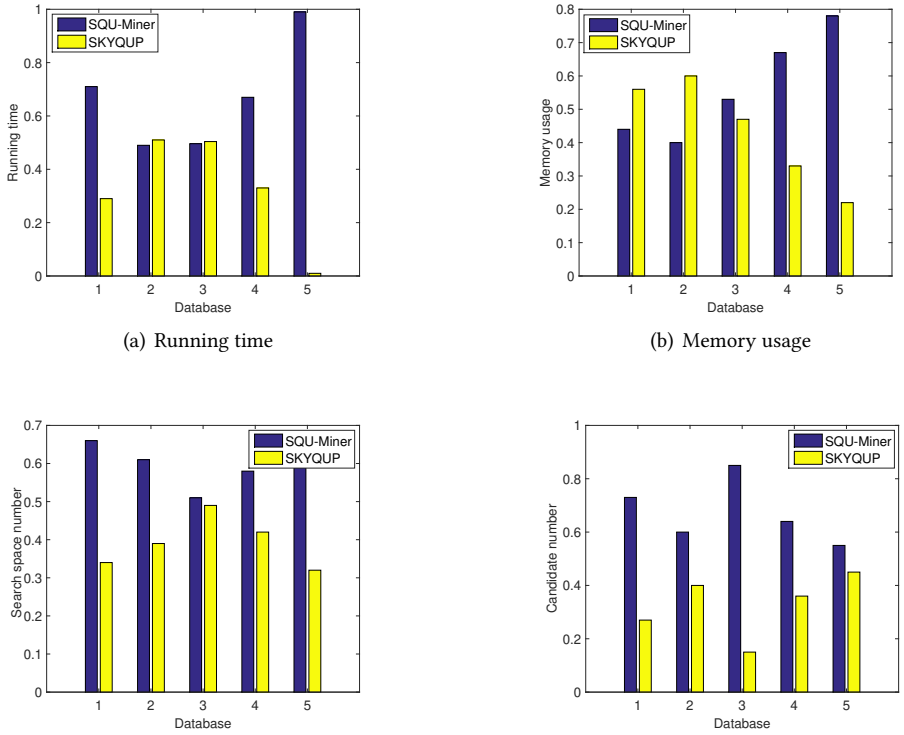
Looking at Table 8, it can be observed that, for retail, accident and chess databases, the second approach saves more memory than the first. For example in accidents database, the proposed SQU-Miner needs 208MB memory while the SKYQUP algorithm only needs 58MB memory and the memory usage of SKYQUP algorithm is up to almost four orders of magnitude smaller than the SQU-Miner algorithm. But for mushroom and foodmart databases, the second approach needs more memory compared to the first approach. However, in terms of running time, the second method is better than the first method. For now, since memory is not valuable to us, and efficiency is increasingly valued, so taken together the SKYQUP algorithm is better than the SQU-Miner algorithm.

6.3 Search space size

Search space size is explored in this section which is used for exploration of SQUPs and is then evaluated using a collection of datasets. We show our strong results in Table 9 and Fig. 4(c).

Table 9. Search space.

Database \ Algorithm	SQU-Miner	SKYQUP
Mushroom	2,466,082	1,254,926
Foodmart	1,879,333	1,212,356
Retail	1,033,394,856	981,229,210
Chess	14,194,845	10,281,155
Accident	-	126,625,389



(a) Running time (b) Memory usage

(c) Search space number (d) Candidate number

Fig. 4. Experimental results in term of running time, memory usage, search space number and candidate number (x-axis indicates the name of the listed database from top to down in Table 6).

SQU-Miner clearly needs a very large search space as shown in Table 9. In the mushroom database as an example, SQU-Miner needs to explore 2, 466, 082 nodes as search space while in contrast SKYQUP explores almost half 1, 254, 926 nodes as search space and the # of search space count of SKYQUP algorithm is up to almost two orders of magnitude smaller than the proposed SQU-Miner algorithm. Whether it is a sparse database or a dense database, the search space of the SKYQUP algorithm is always smaller than the SQU-Miner algorithm. This is reasonable since

the difference between the two different utility-max structures, as the second approach can prune much more search space from the searching tree than the first approach.

6.4 Candidate Size

Table 10. The number of candidate of the two algorithms.

Database \ Algorithm	SQU-Miner	SKYQUP
Mushroom	5264	1970
Foodmart	164	108
Retail	1472	254
Chess	5276	2950
Accident	80	65

In this subsection, the candidate size produced when exploring the SQUPs is explored and given in Table 10 and Fig. 4(d). We can see that the proposed SQU-Miner will produce more candidates while discovering the SQUPs compared to the proposed SKYQUP algorithm from the results in Table 10. For example in the retail database, the SQU-Miner algorithm produces 1,472 candidates while the SKYQUP produces 254 candidates and the number of candidate count of SKYQUP algorithm is up to almost seven orders of magnitude smaller than the SQU-Miner algorithm. Whether it is a sparse database or a dense database, the candidate count of the SKYQUP algorithm is always smaller than the SQU-Miner algorithm. The reason is explained in the first part of this section.

7 CONCLUSION AND FUTURE WORK

FIM and HUIM have been proposed to discover frequent itemsets or high utility itemsets, respectively. However, in practical applications with real-world consequences, there is often more than one aspect that should be considered in the decision-making process. Unfortunately, none of the current methodologies consider both quantity and utility factors when making decisions. In some practical applications, for example in store, salesmen and retailers often pay more attention to the sales volume of goods. In this paper, a new pattern called skyline quantity-utility pattern (SQUP) is proposed and is used to provide better solutions in the decision-making process by considering both quantity and utility together. The designed algorithms use the utility-quantity-list structure to store necessary information contain both quantity and utility for mining SQUPs, which greatly reduces the amount of calculation. Besides, the utility-max structures are designed to keep the maximum utility for each quantity, which greatly reduces the size of the searching space. Lastly, experiments were conducted on several databases and conclusions were achieved based on our experiments. The proposed algorithms show outstanding performance. Thus, they can be very suitable when applied to real-world, real-time application as shown in our introduced examples. For the proposed algorithms applied in the volunteer computing network using a supermarket scenario, the customers' cell phones not only enhance the speed of purchases but also collect necessary information for use in the skyline mining algorithms. It significantly improves the commercial value of the proposed mining algorithms. Due to the lack of research skyline mining combined with volunteer computing, as such we can further explore the expansion of skyline mining from big data, uncertain data or dynamic databases. Furthermore, we can also open the door to explore new skyline patterns considering other factors except for frequency, quantity and utility in future works. In addition, it will become necessary for the implementation of effective structures that can help to discover SQUPs efficiently.

ACKNOWLEDGMENTS

This research was partially funded by the Natural Sciences Research Council of Canada (NSERC) Discovery Grant program (RGPIN-2020-05363) held by Dr. Gautam Srivastava.

REFERENCES

[1] Rakesh Agarwal and Ramakrishnan Srikant. 1994. Fast algorithms for mining association rules. In *International Conference on Very Large Data Bases*. 487–499.
[2] Rakesh Agrawal, Tomasz Imieliński, and Arun Swami. 1993. Mining association rules between sets of items in large databases. In *ACM SIGMOD International Conference on Management of Data*. 207–216.
[3] Usman Ahmed, Jerry Chun-Wei Lin, Gautam Srivastava, Rizwan Yasin, and Youcef Djenouri. 2020. An evolutionary model to mine high expected utility patterns from uncertain databases. *IEEE Transactions on Emerging Topics in Computational Intelligence* (2020).
[4] Stephan Borzsony, Donald Kossmann, and Konrad Stocker. 2001. The skyline operator. In *The International Conference on Data Engineering*. 421–430.
[5] Chee-Yong Chan, HV Jagadish, Kian-Lee Tan, Anthony KH Tung, and Zhenjie Zhang. 2006. Finding k-dominant skylines in high dimensional space. In *ACM SIGMOD International Conference on Management of Data*. 503–514.
[6] Raymond Chan, Qiang Yang, and Yi-Dong Shen. 2003. Mining high utility itemsets. In *IEEE International Conference on Data Mining*. 19–19.
[7] Jan Chomicki, Parke Godfrey, Jarek Gryz, and Dongming Liang. 2003. Skyline with presorting. In *International Conference on Data Engineering*, Vol. 3. 717–719.
[8] Microsoft. Example database foodmart of microsoft analysis services. 2000. http://msdn.microsoft.com/en-us/library/aa217032(SQL.80).aspx.
[9] Philippe Fournier-Viger, Jerry Chun-Wei Lin, Rage Uday Kiran, Yun Sing Koh, and Rincy Thomas. 2017. A survey of sequential pattern mining. *Data Science and Pattern Recognition* 1, 1 (2017), 54–77.
[10] Philippe Fournier-Viger, Jerry Chun-Wei Lin, Bay Vo, Tin Truong Chi, Ji Zhang, and Hoai Bac Le. 2017. A survey of itemset mining. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery* 7, 4 (2017), e1207.
[11] Philippe Fournier-Viger, Cheng-Wei Wu, and Vincent S Tseng. 2012. Mining top-k association rules. In *Canadian Conference on Artificial Intelligence*. 61–73.
[12] Wensheng Gan, Chun-Wei Lin, Philippe Fournier-Viger, Han-Chieh Chao, Vincent Tseng, and Philip Yu. 2019. A survey of utility-oriented pattern mining. *IEEE Transactions on Knowledge and Data Engineering* (2019), 1–1.
[13] Wensheng Gan, Jerry Chun-Wei Lin, Han-Chieh Chao, and Justin Zhan. 2017. Data mining in distributed environment: a survey. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery* 7, 6 (2017), e1216.
[14] Wensheng Gan, Jerry Chun-Wei Lin, Philippe Fournier-Viger, Han-Chieh Chao, and Philip S Yu. 2019. A survey of parallel sequential pattern mining. *ACM Transactions on Knowledge Discovery from Data* 13, 3 (2019), 1–34.
[15] Wensheng Gan, Jerry Chun-Wei Lin, Jiexiong Zhang, and Philip S Yu. 2020. Utility mining across multi-sequences with individualized thresholds. *ACM Transactions on Data Science* 1, 2 (2020), 1–29.
[16] Bart Goethals and MJ Zaki. 2003. Frequent itemset mining implementations repository. http://fimi.cs.helsinki.fi.
[17] Vikram Goyal, Ashish Sureka, and Dhaval Patel. 2015. Efficient skyline itemsets mining. In *The International C* Conference on Computer Science & Software Engineering*. 119–124.
[18] Jiawei Han, Jian Pei, and Yiwen Yin. 2000. Mining frequent patterns without candidate generation. *ACM Sigmod Record* 29, 2 (2000), 1–12.
[19] Donald Kossmann, Frank Ramsak, and Steffen Rost. 2002. Shooting stars in the sky: An online algorithm for skyline queries. In *International Conference on Very Large Data Bases*. 275–286.
[20] Chun-Wei Lin, Tzung-Pei Hong, and Wen-Hsiang Lu. 2011. An effective tree structure for mining high utility itemsets. *Expert Systems with Applications* 38, 6 (2011), 7419–7424.
[21] Jerry Chun-Wei Lin, Lu Yang, Philippe Fournier-Viger, and Tzung-Pei Hong. 2019. Mining of skyline patterns by considering both frequent and utility constraints. *Engineering Applications of Artificial Intelligence* 77 (2019), 229–238.
[22] Xuemin Lin, Yidong Yuan, Qing Zhang, and Ying Zhang. 2007. Selecting stars: The k most representative skyline operator. In *The International Conference on Data Engineering*. 86–95.
[23] Junqiang Liu, Yunhe Pan, Ke Wang, and Jiawei Han. 2002. Mining frequent item sets by opportunistic projection. In *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 229–238.
[24] Mengchi Liu and Junfeng Qu. 2012. Mining high utility itemsets without candidate generation. In *ACM International Conference on Information and Knowledge Management*. 55–64.
[25] Ying Liu, Wei-keng Liao, and Alok Choudhary. 2005. A two-phase algorithm for fast discovery of high utility itemsets. In *Pacific-Asia Conference on Knowledge Discovery and Data Mining*. 689–695.

[26] Jeng-Shyang Pan, Jerry Chun-Wei Lin, Lu Yang, Philippe Fournier-Viger, and Tzung-Pei Hong. 2017. Efficiently mining of skyline frequent-utility patterns. *Intelligent Data Analysis* 21, 6 (2017), 1407–1423.
[27] Dimitris Papadias, Yufei Tao, Greg Fu, and Bernhard Seeger. 2005. Progressive skyline computation in database systems. *ACM Transactions on Database Systems* 30, 1 (2005), 41–82.
[28] Jong Soo Park, Ming-Syan Chen, and Philip S Yu. 1995. An effective hash-based algorithm for mining association rules. *ACM SIGMOD Record* 24, 2 (1995), 175–186.
[29] Vid Podpecan, Nada Lavrac, and Igor Kononenko. 2007. A fast algorithm for mining utility-frequent itemsets. In *International Workshop on Constraint-based Mining and Learning at ECML/PKDD*. 9–20.
[30] Ashok Savasere, Edward Robert Omiecinski, and Shamkant B Navathe. 1995. *An efficient algorithm for mining association rules in large databases*. Technical Report. Georgia Institute of Technology.
[31] Pankaj Singh, Sudhakar Singh, PK Mishra, and Rakhi Garg. 2019. RDD-Eclat: approaches to parallelize eclat algorithm on spark RDD framework. In *International Conference on Computer Networks and Inventive Communication Technologies*. 755–768.
[32] Gautam Srivastava, Jerry Chun-Wei Lin, Matin Pirouz, Yuanfa Li, and Unil Yun. 2020. A pre-large weighted-fusion system of sensed high-utility patterns. *IEEE Sensors Journal* (2020).
[33] Kian-Lee Tan, Pin-Kwang Eng, Beng Chin Ooi, et al. 2001. Efficient progressive skyline computation. In *International Conference on Very Large Data Bases*, Vol. 1. 301–310.
[34] Vincent S Tseng, Bai-En Shie, Cheng-Wei Wu, and S Yu Philip. 2012. Efficient algorithms for mining high utility itemsets from transactional databases. *IEEE Transactions on Knowledge and Data Engineering* 25, 8 (2012), 1772–1786.
[35] Vincent S Tseng, Cheng-Wei Wu, Philippe Fournier-Viger, and S Yu Philip. 2015. Efficient algorithms for mining top-k high utility itemsets. *IEEE Transactions on Knowledge and data engineering* 28, 1 (2015), 54–67.
[36] Vincent S Tseng, Cheng-Wei Wu, Bai-En Shie, and Philip S Yu. 2010. UP-Growth: an efficient algorithm for high utility itemset mining. In *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 253–262.
[37] Jimmy Ming-Tai Wu, Jerry Chun-Wei Lin, and Ashish Tamrakar. 2019. High-utility itemset mining with effective pruning strategies. *ACM Transactions on Knowledge Discovery from Data* 13, 6 (2019), 1–22.
[38] Tsu-Yang Wu, Jerry Chun-Wei Lin, Unil Yun, Chun-Hao Chen, Gautam Srivastava, and Xianbiao Lv. 2020. An efficient algorithm for fuzzy frequent itemset mining. *Journal of Intelligent & Fuzzy Systems* (2020), 1–11.
[39] Hong Yao, Howard J Hamilton, and Cory J Butz. 2004. A foundational approach to mining itemset utilities from databases. In *SIAM International Conference on Data Mining*. 482–486.
[40] Jieh-Shan Yeh, Yu-Chiang Li, and Chin-Chen Chang. 2007. Two-phase algorithms for a novel utility-frequent mining model. In *Pacific-Asia Conference on Knowledge Discovery and Data Mining*. 433–444.
[41] M.J. Zaki, S. Parthasarathy, M. Ogihara, and W Li. 1997. New algorithm for fast discovery of association rules. In *International Conference on Knowledge Discovery and Data Mining*. 283–286.
[42] Mohammed Javeed Zaki. 2000. Scalable algorithms for association mining. *IEEE Transactions on Knowledge and Data Engineering* 12, 3 (2000), 372–390.