

Incremental skyline frequent-utility itemset mining

Xiaojie Zhang^{1,3}, Guoting Chen^{*2}, Linqi Song^{*3}, and Wensheng Gan⁴

¹ Harbin Institute of Technology, Shenzhen, Guangdong 518055, China
xzhang3338-c@my.cityu.edu.hk

² Great Bay University, Dongguan, Guangdong 523000, China
guoting.chen@univ-lille.fr

³ City University of Hong Kong, Hong Kong, China
linqi.song@cityu.edu.hk

⁴ College of Cyber Security, Jinan University, Guangzhou 510632, China
wsgan001@gmail.com

Abstract. High frequency and utility are two crucial objectives in the field of data mining, as they can obtain insightful patterns for decision-making. Most methods of frequent-utility pattern mining are based on thresholds, and they lose itemsets with low frequency but high utility, as well as those with low utility but high frequency. Skyline frequent-utility pattern mining addresses this issue, but the existing ones are designed for static databases. In real-world scenarios, the database usually receives new transactions incrementally. Therefore, designing an efficient algorithm to find skyline frequent-utility patterns in a dynamic database is necessary. This paper proposes an algorithm named ISFUM (Incremental Skyline Frequent-Utility itemset Miner) to realize this purpose. Using global and local utility-lists, the ISFUM algorithm avoids rescanning the original database during incremental mining. New pruning strategies are designed to further conserve computational time by reducing candidate validation. Experimental results demonstrate the efficiency and effectiveness of our algorithm.

Keywords: Frequent-utility pattern · Incremental mining · Skyline pattern · Pruning strategy

1 Introduction

High frequency and utility are crucial objectives in data mining, as they unveil valuable patterns for decision-making. High frequency pattern mining, as exemplified by Apriori [1] and FP-Growth [4], seeks itemsets exceeding a user-defined minimal support threshold (*minsup*). For instance, the {diapers, beer} pattern is renowned for boosting sales. On the other hand, high utility pattern mining [9, 11] aims to discover itemsets maximizing profit, requiring utility not to fall below a user-defined minimal utility threshold (*minutil*). Frequent-utility

* Corresponding authors

pattern mining combines the aspects of high frequency and utility. Traditional frequent-utility pattern mining methods [5] often rely on predefined thresholds (*minsup* and *minutil*), potentially missing itemsets with low frequency but high utility or vice versa. To overcome this limitation, skyline frequent-utility pattern mining [3] was introduced.

In frequent-utility pattern mining, the skyline approaches prove valuable for identifying itemsets that outperform others in frequency or utility. However, existing skyline frequent-utility pattern mining methods are tailored for static databases. Nevertheless, databases are dynamic in practical applications, constantly receiving new transactions. Designing an efficient method for finding skyline frequent-utility itemsets (SFUIs) in a dynamic database is necessary. However, it poses **challenges**: (1) New transaction records may invalidate previous SFUIs. How can we find new SFUIs without rescanning the old data in the database? (2) Exploring a search space to find new SFUIs is essential; how can we leverage old knowledge to enhance pruning in later mining processes?

This paper proposes the ISFUM (Incremental Skyline Frequent-Utility item-set Miner) algorithm for discovering SFUIs in dynamic databases. Using global and local utility-lists, the algorithm avoids rescanning the original database during incremental mining, saving computational time. New pruning strategies capitalize on old SFUIs knowledge to improve efficiency by reducing candidate validation. Experimental results prove the efficiency and effectiveness of our algorithm.

2 Related work

Kung et al. [6] first introduced the skyline concept for identifying ideal points through a divide-and-conquer method. The subsequent development of the skyline operator [2] garnered attention from researchers. SKYMINE [3] first employs the skyline concept in pattern mining to discover SFUIs. It utilizes a utility-pattern tree structure and comprises two stages: the filter phase for candidate generation and the refine stage for candidate verification. Lin et al. [7] proposed a one-phase algorithm based on list structures for skyline frequent-utility item-set mining (SFUIM), which outperforms SKYMINE. SFU-Miner [12] improves [7] by introducing a novel structure called utility-max to enhance search space pruning. SKYFUP-D [8] and SKYFUP-B [8] are one-phase algorithms based on list structures and respectively employ depth-first and breadth-first searches for finding SFUIs. SFUI-UF [13] improves the performance of SFUIM through various filtering methods such as max utility array, transaction weighted utilization, and utility-based filtering. SQU-Miner [15] and SKYQUP [15] focus on identifying skyline quantity-utility patterns. EMSFUI-D [10] and EMSFUI-B [10], adopting depth-first and breadth-first strategies, introduce pruning strategies to facilitate efficient SFUIM. Wu et al. [14] extended their efforts to find skyline quantity-utility patterns in big data environments, proposing three MapReduce tasks for efficient results. SFUP-SP [16], designed for big data environments on the Spark platform, accelerates the process through parallelism and deploys three pruning strategies to minimize the search space.

These diverse approaches collectively contribute to the evolution of SFUIM, each introducing novel strategies or optimizations to enhance the efficiency and applicability of the process. However, there are no methods for SFUIM in dynamic incremental datasets. This paper addresses this issue by proposing an efficient algorithm called ISFUM.

3 Problem formulation

Definition 1 (Incremental transaction database) In an incremental transaction database, new transactions are added over time, comprising two parts in each snapshot: the original database \mathcal{D} and new coming transactions \mathcal{D}^+ . Each transaction T_{tid} is identified by a unique transaction identifier tid and includes tuples of the form (i, q) , where i is an item from a finite set I , and q is the quantity or internal utility $iu(i, T_{tid})$ of the item. Importantly, each item i is associated with an external utility $eu(i)$, representing the unit profit.

Table 1. Incremental transaction database

	tid	items and quantities	external utility
\mathcal{D}	1	$(a, 1), (b, 5), (c, 2), (d, 3), (e, 1), (f, 1)$	5, 2, 1, 2, 3, 5
	2	$(b, 4), (c, 2), (d, 3), (e, 1)$	2, 1, 2, 3
	3	$(a, 1), (c, 2), (d, 2)$	5, 1, 2
	4	$(a, 2), (c, 6), (e, 2), (g, 5)$	5, 1, 3, 1
\mathcal{D}^+	5	$(b, 2), (c, 2), (e, 1), (f, 1), (g, 9)$	2, 1, 3, 5, 1
	6	$(b, 1), (d, 2)$	2, 2

Definition 2 (Frequency and utility) The frequency of an itemset X is defined as the count of transactions that contain X , which is expressed as $F(X) = |\{T_p : X \subseteq T_p\}|$. The utility of an item i within a transaction T_p is the product of its internal and external utility, which is defined as $U(i, T_p) = iu(i, T_p) \times eu(i)$. The utility of an itemset X within a single transaction T_p is computed by summing the utility of all items in X , represented as $U(X, T_p) = \sum_{i \in X} U(i, T_p)$. Extending to a transaction database \mathcal{D} , the utility of X is the summation of its utility across all transactions containing X , denoted as $U(X) = \sum_{T_p \in \mathcal{D} \wedge X \subseteq T_p} U(X, T_p)$. The remaining itemset of X in T_p , denoted as $T_p \setminus X$, is the set of all items in T_p that occur after every item of X . The remaining utility of X in T_p is defined as $RU(X, T_p) = \sum_{i \in T_p \setminus X} U(i, T_p)$. The total remaining utility of X in the transaction database \mathcal{D} is defined as $RU(X) = \sum_{T_p \in \mathcal{D} \wedge X \subseteq T_p} RU(X, T_p)$.

Definition 3 (Skyline frequent-utility itemset) Let X and Y represent two itemsets. We say that X **dominates** Y (or Y is dominated by X) if the frequency and utility of X surpass or equal those of Y and exceed those of Y in at least one dimension. An itemset Z is a **skyline frequent-utility itemset** (SFUI) if no itemset dominates Z .

For example, we consider the itemsets $\{a, c\}$ and $\{c, e\}$ in the transaction database shown in Table 1. In the original database \mathcal{D} , the frequency and utility of $\{a, c\}$ are respectively $F(\{a, c\}) = 3$ and $U(\{a, c\}) = U(\{a, c\}, T_1) + U(\{a, c\}, T_3) + U(\{a, c\}, T_4) = (1 \times 5 + 2 \times 1) + (1 \times 5 + 2 \times 1) + (2 \times 5 + 6 \times 1) = 30$, while for $\{c, e\}$, $F(\{c, e\}) = 3$ and $U(\{c, e\}) = 22$. Since $F(\{a, c\}) \geq F(\{c, e\})$ and $U(\{a, c\}) > U(\{c, e\})$, $\{a, c\}$ dominates $\{c, e\}$. No itemset can dominate $\{a, c\}$ in \mathcal{D} , making $\{a, c\}$ a SFUI in \mathcal{D} . In the new database $\mathcal{D} \cup \mathcal{D}^+$, $F(\{a, c\})$ and $U(\{a, c\})$ remain the same as in \mathcal{D} , while $F(\{c, e\}) = 4$ and $U(\{c, e\}) = 27$. Now, $F(\{c, e\}) > F(\{a, c\})$ and $U(\{c, e\}) < U(\{a, c\})$, meaning that $\{c, e\}$ is no longer dominated by $\{a, c\}$. No itemset can dominate $\{c, e\}$ in $\mathcal{D} \cup \mathcal{D}^+$, making $\{c, e\}$ a new SFUI in $\mathcal{D} \cup \mathcal{D}^+$.

Definition 4 (Search tree) While seeking SFUIs in a transaction database, it is imperative to visit a considerable search space. The search space can be envisioned as a search tree, where each node corresponds to an itemset requiring examination, and each branch signifies an extension operation stemming from the current node. Items are sorted lexicographically to prevent the generation of redundant nodes representing identical itemsets.

Problem description. Given a transaction database \mathcal{D} , there are some methods to find SFUIs. When new transactions \mathcal{D}^+ come in, the state of the transaction database changes. We utilize \mathcal{D}' to represent the new transaction database, $\mathcal{D}' = \mathcal{D} \cup \mathcal{D}^+$. It is time-consuming to utilize the same methods on \mathcal{D}' , this is primarily due to the need to recompute the old portion \mathcal{D} , which is an inefficient use of computational resources. To address this limitation, our objective is to devise an algorithm capable of mining SFUIs in \mathcal{D}' without rescanning the old transaction database \mathcal{D} . In addition, we aim to leverage the old knowledge of SFUIs previously identified in \mathcal{D} to expedite the SFUIM process in \mathcal{D}' .

4 Algorithm

In this section, we present the ISFUM (Incremental Skyline Frequent-Utility itemset Miner) algorithm, suitable for mining SFUIs in incremental transaction databases. We employ superscripts to distinguish between global and local variables, where global variables are accessible to any object without explicitly treating them as input or output, while local variables can only be accessed by specific objects and require explicit treatment as input or output.

4.1 Storage structures

ISU-1 and *ISU-2* are used to store frequency and utility for respectively 1-itemsets and 2-itemsets of the database \mathcal{D} being mined. Both *umax* and *SFUA* are arrays to respectively store utility and SFUIs, with sizes equaling $|\mathcal{D}| + 1$. *umax*[*i*] stores maximal utility among all visited itemsets whose frequencies are no less than *i*, while *SFUA*[*i*] stores SFUIs among all visited itemsets whose

	{b}				{c}				{b,c}		
	<i>tid</i>	<i>U</i>	<i>RU</i>		<i>tid</i>	<i>U</i>	<i>RU</i>		<i>tid</i>	<i>U</i>	<i>RU</i>
	1	10	16		1	2	14		1	12	14
	2	8	11		2	2	9		2	10	9
	<i>F</i> :2	<i>U</i> :18	<i>RU</i> :27		3	2	4	...	<i>F</i> :2	<i>U</i> :22	<i>RU</i> :23
<i>ind</i> →	5	4	19		4	6	11		5	6	17
	6	2	4		<i>F</i> :4	<i>U</i> :12	<i>RU</i> :38		<i>F</i> :1	<i>U</i> :6	<i>RU</i> :17
	<i>F</i> :2	<i>U</i> :6	<i>RU</i> :23	<i>ind</i> →	5	2	17				
					<i>F</i> :1	<i>U</i> :2	<i>RU</i> :17				

Fig. 1. Utility-lists of the database in Table 1.

frequencies are equal to i . Initially, each element $umax[i]$ is 0, and each $SFUA[i]$ is empty. Then $ISU-1$ and $ISU-2$ are used to update $umax$. During the mining process, each search tree node represents a candidate itemset that needs to be verified. If the candidate satisfies the corresponding condition, the $umax$ and $SFUA$ will be updated (refer to EMSFUI-D: judge). As the accessed candidates increase, the states of $umax$ and $SFUA$ change. The updating principle for $umax$ is as follows: once a candidate with frequency F_e and utility U_e such that $U_e > umax[F_e]$ is encountered, we use U_e to update all $umax[i]$ with $i \leq F_e$. In our ISFUM algorithm, $ISU-1^g$, $ISU-2^g$, $umax^g$, and $SFUA^g$ are global variables. Once new transactions \mathcal{D}^+ come in, the frequency and utility for each 1-itemset and 2-itemset in \mathcal{D}^+ are merged to $ISU-1^g$ and $ISU-2^g$, $umax^g$ and $SFUA^g$ experience expansion to make their size become $|\mathcal{D} \cup \mathcal{D}^+| + 1$. The updating principle of $umax$ remains unchanged.

A **utility-list** UL_X is composed of the itemset X it represents, an element array ordered by tid , and the corresponding values $F(X)$, $U(X)$, and $RU(X)$. Each element $E = UL_X[i]$ represents $\langle tid, U(X, T_{tid}), RU(X, T_{tid}) \rangle$. The utility-lists used in ISFUM are illustrated in Figure 1. Each utility-list comprises a global and local part, separated by *ind*. The global part corresponds to the old transaction database \mathcal{D} , while the local part represents the new transaction records \mathcal{D}^+ . After each incremental mining, the local utility-lists will be merged into the global utility-lists. For convenience, we use $sumU(X)$ to represent $U(X) + RU(X)$ and symbol $\&$ to represent the combination of global and local parts.

4.2 Pruning strategies

The following Theorem inspires us to design pruning strategies by utilizing previous knowledge on original \mathcal{D} .

Theorem 1 *Let $S_{\mathcal{D}}$ be SFUIs of \mathcal{D} , F_m and U_m the respectively minimal frequency and utility of $S_{\mathcal{D}}$ in \mathcal{D}^+ , for any itemset X , if its frequency and utility in \mathcal{D}^+ satisfy $F_{\mathcal{D}^+}(X) < F_m$ and $U_{\mathcal{D}^+}(X) < U_m$, then X can not be SFUIs of \mathcal{D}' .*

Proof. If new SFUIs emerge, they must exist in \mathcal{D}^+ since only itemsets in \mathcal{D}^+ with an increase in frequency or utility have the potential to become new SFUIs in \mathcal{D}' . For any itemset X in \mathcal{D}^+ that does not belong to $S_{\mathcal{D}}$ (where $S_{\mathcal{D}}$ represents

Algorithm 1: ISFUM@Main

Input: ULs^g , global utility-lists; $SFUA^g$, an array to store SFUIs of \mathcal{D} ; \mathcal{D}^+ ; new transactions.

Output: $SFUA^g$, an array to store SFUIs of \mathcal{D}' .

- 1 first scan \mathcal{D}^+ to:
- 2 (a) calculate F and U for each 1-itemset and update $ISU-1^g$;
- 3 (b) calculate F and U for SFUIs in $SFUA^g$ and get minimal F_m^g and U_m^g ;
- 4 second scan \mathcal{D}^+ to:
- 5 (a) sort all items in each transaction lexicographically;
- 6 (b) build local utility-lists ULs^l for each 1-itemset;
- 7 (c) calculate F and U for each 2-itemset and update $ISU-2^g$;
- 8 use $ISU-1^g$ and $ISU-2^g$ to update $umax^g$;
- 9 get corresponding global utility-lists ULs^g for local utility-lists ULs^l ;
- 10 call **Core**(null&null, ULs^l & ULs^g);
- 11 merge ULs^l to ULs^g ;

SFUIs of \mathcal{D}), there must be an itemset Y in $S_{\mathcal{D}}$ satisfying $F_{\mathcal{D}}(Y) \geq F_{\mathcal{D}}(X)$ and $U_{\mathcal{D}}(Y) \geq U_{\mathcal{D}}(X)$. As F_m and U_m represent the minimal frequency and utility of $S_{\mathcal{D}}$ in \mathcal{D}^+ , it follows that $F_{\mathcal{D}^+}(Y) \geq F_m$ and $U_{\mathcal{D}^+}(Y) \geq U_m$. If the frequency and utility of X in \mathcal{D}^+ satisfy $F_{\mathcal{D}^+}(X) < F_m$ and $U_{\mathcal{D}^+}(X) < U_m$, then $F_{\mathcal{D}'}(X) = F_{\mathcal{D}}(X) + F_{\mathcal{D}^+}(X) < F_{\mathcal{D}}(Y) + F_{\mathcal{D}^+}(Y) = F_{\mathcal{D}'}(Y)$ and $U_{\mathcal{D}'}(X) = U_{\mathcal{D}}(X) + U_{\mathcal{D}^+}(X) < U_{\mathcal{D}}(Y) + U_{\mathcal{D}^+}(Y) = U_{\mathcal{D}'}(Y)$. This implies that Y dominates X in \mathcal{D}' . Hence, it cannot be a SFUI of \mathcal{D}' .

We derive the following two pruning strategies by referring to Theorem 1 with the same notations.

Pruning strategy 1 For itemset X in \mathcal{D}^+ , if $F_{\mathcal{D}^+}(X) < F_m$ and $sumU_{\mathcal{D}^+}(X) < U_m$, then its extensions cannot be SFUIs of \mathcal{D}' .

Pruning strategy 2 Let P , P_x and P_y be itemsets in \mathcal{D}^+ , where P_x and P_y are extensions of P with item x and y respectively, $diff_{\mathcal{D}^+} = \{T_p \mid T_p \in \mathcal{D}^+ \wedge P_x \subseteq T_p \wedge P_y \subseteq T_p \wedge P_x \neq P_y\}$. If $F_{\mathcal{D}^+}(P_x) - F_{diff_{\mathcal{D}^+}}(P_x) < F_m$ and $sumU_{\mathcal{D}^+}(P_x) - sumU_{diff_{\mathcal{D}^+}}(P_x) < U_m$, then P_{xy} and its extensions cannot be SFUIs of \mathcal{D}' .

4.3 Main process

The input of Algorithm 1 are global utility-lists ULs^g of the original database \mathcal{D} , the array $SFUA^g$ to store SFUIs in \mathcal{D} and the new coming transactions \mathcal{D}^+ . The algorithm first scans \mathcal{D}^+ to calculate F and U for each 1-itemset and update $ISU-1^g$ simultaneously. It also calculates F and U for old SFUIs in $SFUA^g$ in \mathcal{D}^+ and gets minimal F_m^g among all F and minimal U_m^g among all U . It second scans \mathcal{D}^+ to sort all items lexicographically and build local utility-lists ULs^l for each 1-itemset in \mathcal{D}^+ . At the same time, it calculates F and U for each 2-itemset in

Algorithm 2: ISFUM@Core

Input: $UL_P^l \& UL_P^g$, local and global utility-list of P ; $ULs^l \& ULs^g$, local and global utility-lists for extensions of P .

Output: $SFUA^g$, the array to store SFUIs of \mathcal{D}' (or $\mathcal{D} \cup \mathcal{D}^+$).

```

1 for each  $UL_{P_x}^l \& UL_{P_x}^g \in ULs^l \& ULs^g$  do
2   if  $UL_{P_x}^l.F \geq F_m^g \vee UL_{P_x}^l.U \geq U_m^g$  (Theorem 1) then
3     if  $UL_{P_x}^l.U + UL_{P_x}^g.U \geq umax^g[UL_{P_x}^l.F + UL_{P_x}^g.F]$  then
4        $\text{call EMSFUI-D.judge}(UL_{P_x}^l \& UL_{P_x}^g, umax^g, SFUA^g)$ ;
5     end
6   end
7   if  $UL_{P_x}^l.F \geq F_m^g \vee UL_{P_x}^l.sumU \geq U_m^g$  (Strategy 1) then
8     if  $UL_{P_x}^l.U + UL_{P_x}^g.U \geq umax^g[UL_{P_x}^l.F + UL_{P_x}^g.F]$  then
9        $exULs^l = \text{null}$ ;  $exULs^g = \text{null}$ ;
10      for each  $UL_{P_y}^l \& UL_{P_y}^g \in ULs^l \& ULs^g \wedge y \succ x$  do
11         $UL_{P_{xy}}^l \& UL_{P_{xy}}^g = \text{Construct}(UL_P^l \& UL_P^g, UL_{P_x}^l \& UL_{P_x}^g,$ 
12           $UL_{P_y}^l \& UL_{P_y}^g)$ ;
13        if  $UL_{P_{xy}}^l \& UL_{P_{xy}}^g \neq \text{null} \wedge UL_{P_{xy}}^l \& UL_{P_{xy}}^g$  is not empty then
14           $exULs^l \& exULs^g = exULs^l \& exULs^g + UL_{P_{xy}}^l \& UL_{P_{xy}}^g$ ;
15        end
16      end
17       $\text{call Core}(UL_{P_x}^l \& UL_{P_x}^g, exULs^l \& exULs^g)$ ;
18    end
19 end

```

\mathcal{D}^+ and merges them to $ISU-2^g$. Then the algorithm uses $ISU-1^g$ and $ISU-2^g$ to update $umax^g$. For each local utility-list in ULs^l representing one 1-itemset X , the algorithm gets the corresponding global utility-list from ULs^g to represent the same 1-itemset X , and we still denote the corresponding global utility-lists set of ULs^l by ULs^g . Then it calls Algorithm 2 to mine new SFUIs from $\mathcal{D} \cup \mathcal{D}^+$. The new SFUIs are also stored in $SFUA^g$. After this mining process, the local ULs^l are merged to global ULs^g .

Algorithm 2 is the core method for validation and pruning. The input are local and global utility-lists $UL_P^l \& UL_P^g$ of an itemset P , local and global utility-lists $ULs^l \& ULs^g$ for extensions of P . For each $UL_{P_x}^l \& UL_{P_x}^g$ in $ULs^l \& ULs^g$ (including local and global), the algorithm first verifies if P_x is a SFUI and then examine if P_x 's extensions are SFUIs. For the first verification for P_x , if $UL_{P_x}^l.F < F_m^l$ and $UL_{P_x}^l.U < U_m^l$, then P_x cannot be a SFUI. Otherwise, it verifies P_x by the same process as in EMSFUI-D (lines 3 to 5). For the second examination of P_x 's extensions, if $UL_{P_x}^l.F < F_m^l$ and $UL_{P_x}^l.sumU < U_m^l$ (Strategy 1), then there is no need to explore these extensions. Otherwise, the algorithm continues to verify the condition $UL_{P_x}^l.U + UL_{P_x}^g.U \geq umax^g[UL_{P_x}^l.F + UL_{P_x}^g.F]$. If it is not satisfied, then they are pruned.

Algorithm 3: ISFUM@Construct

Input: $UL_P^l \& UL_P^g$, local and global utility-list of P ; $UL_{P_x}^l \& UL_{P_x}^g$, local and global utility-list of P with x ; $UL_{P_y}^l \& UL_{P_y}^g$, local and global utility-list of P with y .

Output: $UL_{P_{xy}}^l \& UL_{P_{xy}}^g$, local and global utility-list for extension of P with $\{x, y\}$.

```

1   $(F^l, U^l, F_t, U_t) = (UL_{P_x}^l.F, UL_{P_x}^l.sumU, UL_{P_x}^l.F + UL_{P_x}^g.F,$ 
    $UL_{P_x}^l.sumU + UL_{P_x}^g.sumU);$ 
2  for  $i = 1, j = 1; i \leq |UL_{P_x}^l| \wedge j \leq |UL_{P_y}^l|$  do
3     $(E_x, E_y) = (UL_{P_x}^l[i], UL_{P_y}^l[j]);$ 
4    if  $E_x.tid == E_y.tid$  then
5      if  $UL_P^l \neq \text{null} \wedge \exists E \in UL_P^l \text{ s.t. } E.tid == E_x.tid$  then
6         $E_{xy} = \langle E_x.tid, E_x.sumU - E.U, E_y.RU \rangle;$ 
7      else
8         $E_{xy} = \langle E_x.tid, E_x.sumU, E_y.RU \rangle;$ 
9      end
10     append  $E_{xy}$  to  $UL_{P_{xy}}^l$ ;
11      $i++; j++;$ 
12   else if  $E_x.tid > E_y.tid$  then
13      $j++;$ 
14   else if  $E_x.tid < E_y.tid$  then
15      $(F^l, U^l, F_t, U_t) = (F^l - 1, U^l - (E_x.sumU), F_t - 1, U_t - (E_x.sumU));$ 
16     if  $(F^l < F_m^g \wedge U^l < U_m^g)$  (Strategy 2)  $\vee (U_t < umax^g[F_t])$  then
17       return null&null;
18     end
19      $i++;$ 
20 end
21 get  $UL_{P_{xy}}^g$  from  $UL_P^g, UL_{P_x}^g$ , and  $UL_{P_y}^g$ ;
22 return  $UL_{P_{xy}}^l \& UL_{P_{xy}}^g$ ;

```

Algorithm 3 is responsible for list combination and is called by Algorithm 2. The input are local and global utility-lists $UL_P^l \& UL_P^g$ of an itemset P , local and global utility-lists $UL_{P_x}^l \& UL_{P_x}^g$ of P with item x , and local and global utility-lists $UL_{P_y}^l \& UL_{P_y}^g$ of P with item y . The output are local and global utility-lists $UL_{P_{xy}}^l \& UL_{P_{xy}}^g$ of P with $\{x, y\}$. The algorithm first traverses the local $UL_{P_x}^l$ and $UL_{P_y}^l$ to construct $UL_{P_{xy}}^l$ (lines 2 to 20). In this process, i and j respectively points to the first element E_x of $UL_{P_x}^l$ and E_y of $UL_{P_y}^l$. If they have the same tid ($E_x.tid == E_y.tid$), the algorithm creates an element E_{xy} and appends it to $UL_{P_{xy}}^l$. The value of $E_{xy}.U$ depends on if UL_P^l contains the same tid (lines 5-9). If $E_x.tid > E_y.tid$, j points to the next element, and the algorithm gets into the next round to compare old E_x and new E_y . If $E_x.tid < E_y.tid$, it updates F^l , U^l , F_t , and U_t values, and checks whether $F^l < F_m^g \wedge U^l < U_m^g$ (Strategy 2) or $U_t < umax^g[F_t]$. If it is satisfied, P_{xy} and its extensions are pruned by returning

Table 2. Characteristics of the datasets

Dataset	$ \mathcal{D} $	$ \mathcal{I} $	AvgLen	Density	Utility	Timestamp
foodmart	4,141	1,559	4.42	0.28%	real	synthetic
ecommerce	14,975	3468	11.71	0.34%	real	real
fruitnut	181,970	1265	3.58	0.28%	real	real
kosarak	990,002	41,270	8.1	0.02%	synthetic	synthetic

null&null. If it is not satisfied, i points to the next element and the algorithm gets into the next round to compare new E_x and old E_y . Similarly, the algorithm next traverse the global $UL_{P_x}^g$ and $UL_{P_y}^g$ to construct $UL_{P_{xy}}^g$ (line 21).

5 Experiment

This section presents a series of experiments conducted on diverse datasets to demonstrate the efficiency and effectiveness of our algorithm. We compare thoroughly with the state-of-the-art skyline frequent-utility algorithm in the static datasets, namely, EMSFUI-D [10]. By examining various indicators, including runtime consumption, memory usage, candidate&joint counts, and patterns across different data sizes and features, we aim to elucidate the distinctive properties of our proposed algorithm.

The baseline algorithm. EMSFUI-D [10] employs depth-first search to systematically explore the search space of all itemsets. The algorithm incorporates two pruning strategies to effectively constrain the search space. To enhance mining performance, EMSFUI-D introduces the ISU-1 and ISU-2 structures. These structures serve to maintain more precise utility upper bounds by preserving the support and utility information for all 1-itemsets and 2-itemsets, respectively.

Datasets and experimental settings. To demonstrate the effectiveness of our methods, we selected 4 datasets, each characterized by distinct features. The codes of our algorithm and the baseline can be found in our GitHub repository⁵. These datasets are obtained from the SPMF platform, and their features are outlined in Table 2. Key attributes include $|\mathcal{D}|$, indicating the total number of transactions in \mathcal{D} ; $|\mathcal{I}|$, representing the count of distinct items in the dataset; **AvgLen**, denoting the average transaction length; and **Density**, calculated as **AvgLen** divided by $|\mathcal{I}|$. Additionally, **Utility** and **Timestamp** signify whether the values in the datasets are real or synthetic.

Runtime analysis. The runtime consumption under different data sizes is shown in Figure 2. We find ISFUM almost consumes less time than EMSFUI-D in all datasets. As the data size increases, this divergence becomes more and more apparent. In particular, the ISFUM algorithm saves one order of magnitude time than EMSFUI-D in the kosarak dataset. Furthermore, we observe that the runtime of both algorithms does not consistently increase with the data scale; occasionally, they exhibit a declining trend. The emergence timing of

⁵ <https://github.com/homexiaojie888/ISFUM.git>

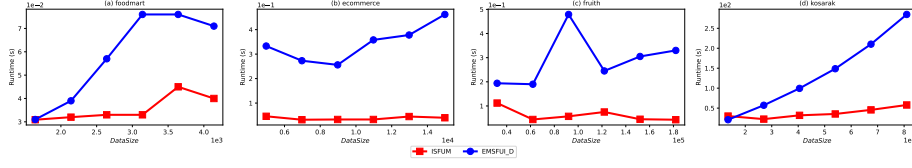


Fig. 2. Runtime when varying *DataSize*.

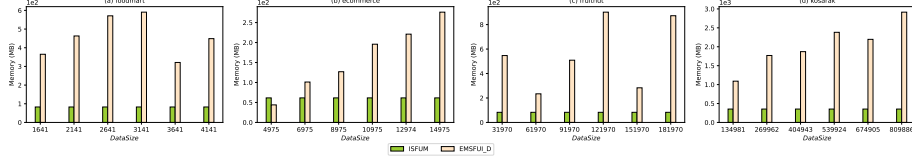


Fig. 3. Memory when varying *DataSize*.

specific perfect candidates primarily leads to this phenomenon, and premature appearance can remove useless candidate nodes early.

Memory analysis. The memory usage under different data sizes is depicted in Figure 3. ISFUM consistently consumes less memory than EMSFUI-D across all datasets. This efficiency is attributed to ISFUM focusing only on the new data, while EMSFUI-D processes both old and new data, leading to higher memory usage. In almost all datasets, ISFUM achieves a memory saving of one order of magnitude compared to EMSFUI-D. We observe that memory usage does not consistently follow an increasing trend with the data scale. This variability is influenced by the garbage collection mechanism of the Java virtual machine (JVM).

Number of candidates and joint counts. To illustrate the efficiency of our algorithm, we compare the number of candidates and joint counts under different data sizes, as shown in Figure 4. For the first *DataSize* in all datasets, ISFUM and EMSFUI-D consistently exhibit the same candidate and joint counts. This is expected because ISFUM and EMSFUI-D are essentially the same process when the old data is empty for ISFUM. We also find that ISFUM visits fewer candidates and experiences fewer joint counts than EMSFUI-D in most cases. This contributes to pruning strategies 1 and 2 lessing down redundant candidates.

Pattern analysis. To demonstrate the effectiveness of our algorithm, we compare the patterns mined by ISFUM and EMSFUI-D across various data sizes, as shown in Figure 5. In each sub-figure, the initial column indicates the dataset sizes, while the subsequent columns display the generated patterns with their corresponding F and U values. We find ISFUM and EMSFUI-D always have the same SFUIs for different data sizes in all datasets; this proves the effectiveness of our ISFUM. We also observe that SFUIs remain unchanged in

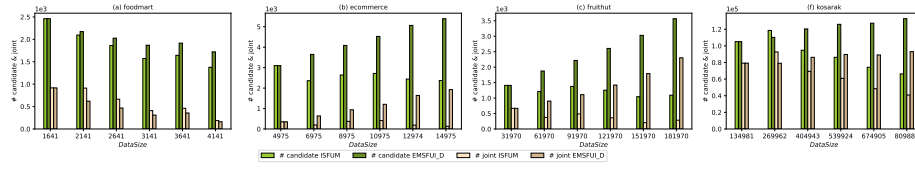


Fig. 4. Candidate and joint number when varying *DataSize*.

foodmart	EMSFUI-D ISFUM		ecommerce		EMSFUI-D	ISFUM	fruitnut		EMSFUI-D	ISFUM	kosarak	EMSFUI-D	ISFUM
1641	[225] : 10, 12059		4975	[22423] : 447, 4155664			31970	[2010] : 6891, 1386433			134981	[6, 11] : 43955, 2419366	
	[1426] : 11, 11968			[8512311] : 499, 2218160				[6] : 81949, 1806248					
	[728] : 12, 7749												
2141	[272] : 13, 14858		6975	[22423] : 592, 5087100			61970	[2010] : 14111, 2837342			269962	[6, 11] : 88069, 4848460	
	[219] : 14, 10332			[8512311] : 663, 3029706				[6] : 163862, 3613596					
2641	[272] : 15, 17595		8975	[22423] : 729, 6084600			91970	[2010] : 21370, 4294420			404943	[6, 11] : 132312, 7290490	
	[918] : 16, 15147			[8512311] : 807, 3668361				[6] : 246108, 5426140					
3141	[272] : 17, 19941		10975	[22423] : 849, 7046655			121970	[2010] : 28531, 5738762			539924	[6, 11] : 176881, 9744288	
	[1373] : 18, 18720			[8512311] : 912, 4274916				[6] : 328332, 7236724					
3641	[304] : 19, 10583		12974	[22423] : 946, 7886745			151970	[2010] : 36423, 7326185			674905	[6, 11] : 220928, 12165782	
	[988] : 15, 20938			[8512311] : 996, 4747356				[6] : 410267, 9041704					
4141	[1292] : 19, 20313		14975	[22423] : 1033, 8560695			181970	[2010] : 43227, 8695504			809886	[6, 11] : 265182, 14596530	
	[1012] : 22, 19034			[8512311] : 1104, 5406791				[6] : 492132, 10840224					
4141	[1373] : 25, 25560												

Fig. 5. Patterns when varying *DataSize*.

the ecommerce, fruitnut, and kosarak datasets while they experience change each time in the foodmart dataset.

6 Conclusions

In this paper, we present an efficient incremental mining algorithm, ISFUM, to extract SFUIs from dynamic databases. Notably, the algorithm showcases the capability to avoid rescanning the original database, leading to significant computational time savings in incremental mining scenarios. A key feature of our approach is the introduction of pruning strategies aimed at optimizing candidate validation. The experimental results demonstrate the efficiency and effectiveness of our algorithm.

References

1. Agrawal, R., Srikant, R.: Fast algorithms for mining association rules in large databases. In: Proc. Intern. Conf. Very Large Data Bases. pp. 487–499 (1994)
2. Börzsönyi, S., Kossmann, D., Stocker, K.: The skyline operator. In: Proc. 17th Intern. Conf. on Data Engineering. pp. 421–430 (2001)
3. Goyal, V., Sureka, A., Patel, D.: Efficient skyline itemsets mining. In: Proc. 8th Intern. Conf. on Computer Science & Software Engineering. pp. 119–124 (2015)
4. Han, J., Pei, J., Yin, Y., Mao, R.: Mining frequent patterns without candidate generation: A frequent-pattern tree approach. Data Min. Knowl. Discov. **8**(1), 53–87 (2004)

5. Kiran, R.U., Reddy, T.Y., Fournier-Viger, P., Toyoda, M., Reddy, P.K., Kitsuregawa, M.: Efficiently finding high utility-frequent itemsets using cutoff and suffix utility. In: Proc. Advances in Knowledge Discovery and Data Mining. Lecture Notes in Computer Science, vol. 11440, pp. 191–203 (2019)
6. Kung, H.T., Preparata, F.P., Patel, D.: On finding the maxima of a set of vectors. *Journal of the ACM* pp. 502–513 (1975)
7. Lin, J.C., Yang, L., Fournier-Viger, P., Dawar, S., Goyal, V., Sureka, A., Vo, B.: A more efficient algorithm to mine skyline frequent-utility patterns. In: Proc. 10th Intern. Conf. on Genetic and Evolutionary Computing. Advances in Intelligent Systems and Computing, vol. 536, pp. 127–135 (2016)
8. Lin, J.C., Yang, L., Fournier-Viger, P., Hong, T.: Mining of skyline patterns by considering both frequent and utility constraints. *Eng. Appl. Artif. Intell.* **77**, 229–238 (2019)
9. Liu, M., Qu, J.: Mining high utility itemsets without candidate generation. In: Intern. Conf. on Information and Knowl. Management. pp. 55–64 (2012)
10. Liu, X., Chen, G., Zuo, W.: Effective algorithms to mine skyline frequent-utility itemsets. *Eng. Appl. Artif. Intell.* **116**, 105355 (2022)
11. Liu, Y., Liao, W., Choudhary, A.N.: A two-phase algorithm for fast discovery of high utility itemsets. In: Proc. Advances in Knowl. Discovery and Data Mining. Lecture Notes in Computer Science, vol. 3518, pp. 689–695 (2005)
12. Pan, J., Lin, J.C., Yang, L., Fournier-Viger, P., Hong, T.: Efficiently mining of skyline frequent-utility patterns. *Intell. Data Anal.* **21**(6), 1407–1423 (2017)
13. Song, W., Zheng, C., Fournier-Viger, P.: Mining skyline frequent-utility itemsets with utility filtering. In: Proc. 18th Pacific Rim Intern. Conf. on Artificial Intelligence. Lecture Notes in Computer Science, vol. 13031, pp. 411–424 (2021)
14. Wu, J.M., Li, R., Lin, J.C.: Skyline pattern mining by quantity-utility constraints in large-scale databases. In: Proc. IEEE Intern. Conf. on Data Mining Workshops. pp. 547–552 (2022)
15. Wu, J.M., Teng, Q., Srivastava, G., Pirouz, M., Lin, J.C.: The efficient mining of skyline patterns from a volunteer computing network. *ACM Trans. Internet Techn.* **21**(4), 89:1–89:20 (2021)
16. Wu, J.M., Zhou, H., Lin, J.C., Srivastava, G., Baza, M.: Mining skyline patterns from big data environments based on a spark framework. *J. Grid Comput.* **21**(2), 22 (2023)