Information Sciences 564 (2021) 1–26

Contents lists available at ScienceDirect

Information Sciences

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

ILUNA: Single-pass incremental method for uncertain frequent pattern mining without false positives

Razieh Davashi [a,b,*]

[a] Faculty of Computer Engineering, Najafabad Branch, Islamic Azad University, Najafabad, Iran
[b] Big Data Research Center, Najafabad Branch, Islamic Azad University, Najafabad, Iran

A R T I C L E   I N F O

Article history:
Received 8 June 2020
Received in revised form 15 February 2021
Accepted 18 February 2021
Available online 27 February 2021

Keywords:
Mining frequent patterns
Uncertain data
Incremental mining
Uncertain frequent patterns
Dynamic data

A B S T R A C T

Nowadays, due to the mass production of uncertain data, numerous methods have been proposed for mining frequent patterns from uncertain data; however, none of them are proper for dynamic data environments. In many real-world applications, transactions are constantly being updated. After incremental updates, the validity of the uncertain patterns changes. The existing static algorithms to handle this state have to rerun the whole mining process from scratch, which is very costly. Incremental-CUF-growth is a method dealing with dynamic data but it generates many false positives and requires an additional time-consuming database scan to filter them. To handle these drawbacks, in this paper, an efficient single-pass method called ILUNA is proposed for incremental mining of uncertain frequent patterns without false positives. It introduces two new data structures namely IUP-List and ICUP-List to efficiently store data which can be increased. Upon receiving each new database, it only updates the lists without having to rebuild them from scratch. This is the first study in which single-pass incremental mining of uncertain frequent patterns is performed. Comprehensive experimental results show that the proposed method dramatically reduces the runtime and enhances the scalability compared to the state-of-the-art methods for dense and sparse incremental datasets.

1. Introduction

Frequent pattern mining is one of the most fundamental fields of data mining which is widely applied in various fields such as marketing, business, medical treatment, networks, industry, finance, and so on. Hence, for more than two decades, a large number of studies have been carried out on mining closed frequent patterns [1], maximal frequent patterns [2], weighted frequent patterns [3,4], sequential frequent patterns [5], constrained frequent patterns [6], high-utility frequent patterns [7], frequent graph patterns [8,9], erasable frequent patterns [10,11], Top-k frequent patterns [12,13], and frequent patterns over data streams [14]. Apriori [15] and FP-growth [16] are regarded as fundamental methods for mining frequent patterns. The Apriori with a level-wise structure suffers from two major problems of mass production of candidates and frequent database scans. Therefore, the FP-growth algorithm was suggested which is a tree structure that extracts patterns using only two database scans without the need to generate candidates. However, this algorithm is inefficient in incremental environments where the database is constantly updated. Therefore, to improve the efficiency of the FP-growth algorithm, numerous studies have been conducted on incremental mining [17–21] of frequent patterns. However, all of these

* Address: Faculty of Computer Engineering, Najafabad Branch, Islamic Azad University, Najafabad, Iran.
E-mail address: davashi@sco.iaun.ac.ir

https://doi.org/10.1016/j.ins.2021.02.067
0020-0255/© 2021 Elsevier Inc. All rights reserved.

algorithms handle precise data, while at the present time, uncertain data are growing quickly due to the technological advances in hardware and data collection methods in real-world applications. Hence, the demands to discover the knowledge of these data have been increased and frequent pattern mining from uncertain data has become an important research topic for researchers. In precise databases, the occurrence probability of each item in the transaction is 1 or 0 while in uncertain databases, each item in the transaction is paired with a unique existential probability value between (0, 1]. Therefore, mining frequent patterns from uncertain data is much more difficult than mining such patterns from precise data. The U-Apriori level-wise approach [22] is the first proposed algorithm for mining uncertain frequent patterns. It has inherited the inefficient nature of the Apriori-based algorithms. Therefore, UF-growth algorithm [23,24] as a tree-based algorithm was proposed that does not have problems of Apriori-based algorithms. However, it has a large size which requires a great volume of memory to hold its tree and much time to extract patterns. To improve the efficiency of U-Apriori and UF-growth algorithms, numerous algorithms such as Apriori-based, H-mine-based, Eclat-based, FP-growth-based, and list-based algorithms were proposed for mining frequent patterns from uncertain data. However, most of these algorithms support static uncertain pattern mining; while, in real-world applications, new databases can be added frequently and continually to the original database. Therefore, dynamic mining is more functional than static mining. In general, uncertain pattern mining algorithms can be divided into two categories namely exact-based and upper bound-based methods. In the exact-based methods, the exact existential probability of items is used to extract frequent patterns but in the upper bound-based methods, the upper bound of the existential probability of items is considered. Therefore, the upper bound-based methods may produce false positives that need to be filtered out with an additional scan. Incremental-CUF-growth algorithm [25] is an upper bound-based algorithm for dealing with dynamic data. It is a modified version of CUF-growth algorithm [26] that produces a lot of false positives due to the loose upper bound on the expected support. It captures uncertain frequent patterns by two scans of the dynamic database, where the second scan is a very time-consuming scan for filtering the false positives from the result set. In general, upper bound-based algorithms are not very suitable for incremental mining because every time a new database is added, they have to scan the new database and rescan the entire database (original + new) for calculating the exact expected support of the patterns. In fact, in incremental environments where data are constantly being added, these algorithms increase the disk input/output cost and lack adequate flexibility. In contrast, exact-based algorithms do not require this additional scan. However, none of the existing exact-based algorithms have incremental mining capability. Indeed, after incremental updates, the validity of the uncertain frequent patterns changes. In this situation, the only way for existing exact-based algorithms is to rerun the whole mining process from scratch which is very time-consuming, especially in large databases. LUNA algorithm [27] is one of the state-of-the-art exact-based algorithms with a new list data structure that extracts frequent patterns from uncertain data without false positives. However, it is inefficient for dynamic environments and whenever a new database is added, it has to rescan the entire database (original + new) twice and repeat the whole process of constructing the UP-Lists and CUP-Lists from scratch. These drawbacks are the main issues of performance degradation. Motivated by the problems mentioned above, an efficient single-pass method for incremental mining frequent patterns from uncertain data is proposed in this paper. Key contributions of this paper are as follows:

- To the best of the researcher's knowledge, this is the first study to address the issue of incremental mining of uncertain frequent patterns with a single database scan;
- A single-pass method ILUNA (**I**ncremental **L**ist-based **UN**certain frequent pattern mining **A**lgorithm) is proposed for incremental mining of uncertain frequent patterns efficiently;
- Two new data structures based on the list named IUP-List and ICUP-List are suggested to efficiently store and maintain data which can be increased;
- Techniques are proposed to eliminate additional calculations of lists processing;
- The proposed method extracts exact results of uncertain frequent pattern mining without any false positives from dynamic databases; and
- Extensive experimental results show the superiority of the ILUNA algorithm.

The rest of this paper is organized as follows: the related works are reviewed in Section 2; Section 3 explains the problem statement and details of the proposed method; Section 4 analyzes the time complexity of the proposed method; the performance evaluation and related discussions are presented in Section 5; and Section 6 offers the conclusion and suggestions for future works.

2. Related works

In recent years, many effective methods [28–30] have been put forward to improve the performance of algorithms for mining frequent patterns from precise data. Due to the massive amount of uncertain data generated by many applications such as mobile or Internet of Things (IoT) devices [31], the proposed methods related to frequent pattern mining from uncertain data have received more attention. Generally, the methods for mining uncertain frequent patterns can be classified as Apriori-based, Eclat-based, H-mine-based, FP-growth-based, and list-based methods. The U-Apriori [22] was the first method devised to deal with uncertain data which had problems of level-wise approaches and was not scalable well for big data. Other Apriori-based algorithms, UCP-Apriori [32], MBP [33], and IMBP [34] also suffer from common drawbacks of the generate-and-test approaches; candidate generation and repeated database scans. Eclat-based methods U-Eclat

[35], UEclat [36], UV-Eclat [37], and U-VIPER [38] are based on the framework of the Eclat algorithm [39] that employ vertical data structure for mining uncertain frequent patterns. H-mine-based approaches such as UH-Mine [40] and P-HMine [41] apply a hyperlinked structure; therefore, they do not have a compact data structure. FP-growth-based methods adopt a tree structure that usually has better performance than previous methods. The UF-growth [23,24] is the first tree-based algorithm that extracts frequent patterns from uncertain data by two database scans. Indeed, it falls into the category of exact-based methods and only allows sharing nodes of each path in UF-tree with the same item name and existential probability value. For this reason, UF-tree may have a large size which requires too much memory for maintaining nodes and long runtime for processing them. Therefore, UFP-growth algorithm [40] was proposed to reduce the size of UF-tree. It is an upper bound-based algorithm that needs two scans for constructing UFP-tree and an extra scan for filtering false positives. Although this algorithm has been designed to reduce the size of the UF-tree, because it depends on the clustering parameter, its size may be as large as an UF-tree. To address problems of the UF-tree and UFP-tree, two upper bound-based algorithms namely CUF-growth and CUF-growth* [26] were proposed. These two algorithms have a compact tree structure, but they produce a large number of false positives due to having a loose upper bound on expected support. Thus, the third scan of the database to eliminate the false positives is very time-consuming. Subsequently, numerous upper bound-based algorithms such as PUF-growth [42], Disc-growth [43], BLIMP-growth [44], TPC-growth [45], and MUF-growth [46] were proposed to tighten the upper bound and reduce the number of false positives. However, these algorithms need three scans of the database and still generate many false positives, especially for low minSups. Besids the UF-tree, two other exact-based methods CUFP-Mine [47] and AT-mine [48] were proposed for mining frequent patterns from uncertain data. The CUFP-Mine generates all supersets for each of its tree nodes and calculates the expected support of each superset which is very time-consuming. In addition, it requires a great volume of memory for maintaining all supersets in the tree nodes. To overcome these problems, the AT-mine was suggested that stores all probability values of each tree path in a separate array named ProArr and its related information on the tail node of the same path. Therefore, the AT-mine requires a great volume of memory for maintaining ProArr and information on the tail nodes. It also needs more time for calculating the total expected support of patterns through multiple references to the ProArr. To improve the performance of the CUFP-Mine and AT-Mine algorithms, LUNA method [27] with a new list-based structure was proposed. It stores information of transactions in UP-Lists and creates CUP-Lists by combining the UP-Lists or CUP-Lists for mining frequent patterns. It scans the database twice to construct UP-Lists. In the first scan, it computes the expected support of all items, deletes the infrequent items, and creates an UP-List for each frequent item. In the second scan, it updates the tuple sets and maximum value of existential probabilities of UP-Lists. Then, in the mining process, it recursively generates the CUP-Lists. Although the LUNA algorithm efficiently succeeds in extracting patterns, like all the above mentioned methods, it lacks the incremental updating ability. In the incremental environments, by receiving any new database, even if a small change occurs, the LUNA has to rescan the updated database (original database + new database) twice and construct the UP-lists and CUP-Lists from the beginning which are not cost-effective. In [25], Incremental-CUF-growth algorithm has been proposed to support incremental mining of uncertain frequent patterns. By receiving any new database, this upper bound-based algorithm scans the new database, reconstructs the original tree branches, and adds new transactions into the original tree structure. Then, it extracts the potential patterns from the tree from scratch and filters the false positives by rescanning the entire database (original + new). As a result, this algorithm has serious drawbacks including the need to rescan the entire database and generate many false positives. Hence, it lacks appropriate performance in dynamic environments. Table 1 illustrates the overall characteristics of FP-growth-based and list-based algorithms of mining uncertain frequent patterns.

To handle incremental mining of uncertain frequent patterns by only a single database scan, a new method will be proposed in the following section of this paper. Since the findings show that the LUNA algorithm is the most efficient approach among investigated methods, the proposed algorithm in this paper improves it.

Table 1
Characteristics of FP-growth-based and list-based algorithms of mining uncertain frequent patterns.

Algorithm	N. of scans	Method	Structure	Ex. of false positives	Incremental capability
UF-Growth [23,24]	2	FP-growth-based	Uncompressed tree	N	N
UFP-Growth [40]	3	FP-growth-based	Uncompressed tree	Y	N
CUF-Growth [26]	3	FP-growth-based	Compact tree	Y	N
CUF-Growth* [26]	3	FP-growth-based	Compact tree	Y	N
PUF-Growth [42]	3	FP-growth-based	Compact tree	Y	N
Disc-Growth [43]	3	FP-growth-based	Compact tree	Y	N
BLIMP-Growth [44]	3	FP-growth-based	Compact tree	Y	N
TPC-Growth [45]	3	FP-growth-based	Uncompressed tree	Y	N
MUF-growth [46]	3	FP-growth-based	Compact tree	Y	N
CUFP-Mine [47]	2	FP-growth-based	Compact tree	N	N
AT-Mine [48]	2	FP-growth-based	Compact tree	N	N
LUNA [27]	2	List-based	Compact structure	N	N
Incremental-CUF-Growth [25]	2	FP-growth-based	Compact tree	Y	Y
Proposed ILUNA	1	List-based	Compact structure	N	Y

3. Proposed method

In this section, the problem statement and some essential concepts related to uncertain frequent pattern mining and incremental mining are introduced. Thereafter, the details of the proposed ILUNA method are described. They include new data structures, pattern mining technique, and additional performance improving techniques along with their examples for incremental uncertain frequent pattern mining.

3.1. Problem statement and definitions

In this section, the concepts of mining frequent patterns from uncertain data are described and then the statement of the problem is clearly stated. Let UDB= $\{T_1, T_2, \ldots, T_n\}$ be an uncertain database containing n transactions and I= $\{x_1, x_2, \ldots, x_m\}$ be a set of distinct items. Then, each transaction T_j in UDB is a subset of I. Table 2 is an example of a transactional uncertain database. The X=$\{x_1, x_2, \ldots, x_k\}$ is a k-itemset or a pattern composed of items belonging to I. If λ be a user predefined minimum support threshold and |UDB| be the size of the UDB, then the minimum Support (minSup) is calculated as minSup=|UDB|$\times\lambda$.

Definition 1. The existential probability value of item x_i in transaction T_j is denoted as $P(x_i, T_j)$

$(0 < P(x_i, T_j) \leq 1)$.

For example, in Table 2:

P(B, TID:100) = 0.1, P(A, TID:400) = 0.9, and P(D, TID:700) = 0.8.

Definition 2. The existential probability of pattern X in transaction T_j is denoted as $P(X, T_j)$ and is defined by:

$$P(X, T_j) = \Pi_{x \epsilon X} P(x, T_j) \tag{1}$$

For example, in Table 2:
P ({A, C}, TID:100) = 0.8 $\times$ 0.2 = 0.16, P ({A, B, D}, TID:600) = 0.5 $\times$ 0.5 $\times$ 0.9 = 0.225.

Definition 3. The expected support of pattern X in the uncertain database is donated as expSup(X) and is defined by:

$$expSup(X) = \sum_{j=1}^{|UDB|} (\Pi_{x \epsilon X} P(x, T_j)) \tag{2}$$

For example, in Table 2:
expSup(A, B) = P({A, B}, TID:100) + P({A, B}, TID:200) + P({A, B}, TID:400) + P({A,B}, TID:600) = (0.8 $\times$ 0.1) + (0.5 $\times$ 0.1) + (0.9 $\times$ 1) + (0.5 $\times$ 0.5) = 1.28.

The expSup(X) satisfies the downward closure property [15] that reduces the search space without any pattern loss as expSup(X) $\leq$ expSup(Y) for all Y $\subset$ X. Therefore, Property 1 can be defined as follows:

Property 1. For any Y $\subset$ X , the expected support satisfies the downward closure property:

(a) If expSup(X) $\geq$ minSup, it is also true that expSup(Y) $\geq$ minSup, and
(b) If expSup(Y) < minSup, it is also true that expSup(X) < minSup.

Table 2
Example of an uncertain database.

	TID	Items				
		A	B	C	D	E
Original database (UDB_0)	100	0.8	0.1	0.2	–	–
	200	0.5	0.1	0.1	–	–
	300		–	0.1	–	–
new database (UDB_1)	400	0.9	1	–	–	0.9
	500	–	–	–	0.2	0.5
new database (UDB_2)	600	0.5	0.5	–	0.9	–
	700	0.4	–	0.6	0.8	–

Proof. Since Y is a subset of X, Y must be present in each transaction containing the X, while its opposite is not true. Therefore, the number of transactions containing X is equal to or less than the number of transactions containing Y. Hence, $\mathrm{expSup}(X)=\sum_{j=1}^{n} P(X,T_j) \le \sum_{j=1}^{n} P(Y,T_j) = \mathrm{expSup}(Y)$ for all n transactions in UDB. If X is a frequent pattern (i.e., minSup $\le$ expSup(X)), then Y also is frequent pattern because minSup$\le$ expSup(X)$\le$expSup(Y). Conversely if Y is an infrequent pattern (i.e., expSup(Y)<minSup), then X is infrequent because expSup(X) $\le$ expSup(Y) $<$ minSup.

Definition 4. A pattern X in UDB is Uncertain Frequent Pattern (UFP) if:

$$\mathrm{expSup}\ (X) \le \mathrm{minSup} \tag{3}$$

Problem statement. This paper focuses on mining uncertain frequent patterns from incremental databases where new databases are continuously added. Indeed, an original uncertain database is a given database at the beginning of the mining process and incremental databases refer to data that are accumulated with the passage of time. Given original uncertain database UDB_0, incremental uncertain database UDB_i, and minSup, consider the S_{UDB_o} refers to the set of uncertain frequent patterns having expSup greater than or equal to minSup value in UDB_0. The updated database is shown by UDB (UDB_0 +- UDB_i) and the expSup of a pattern X in UDB is shown by $\mathrm{expSup}_{UDB}(X)$ which is equal to $\mathrm{expSup}_{UDB_o}(X) + \mathrm{expSup}_{UDB_i}(X)$. The problem of updating uncertain association rules lies in finding the new set of uncertain frequent patterns in UDB S_{UDB} efficiently.

3.2. The overall architecture of the proposed method

Fig. 1 shows the overall architecture of the proposed method named ILUNA. The regular flow of the proposed algorithm is as follows. The ILUNA algorithm constructs IUP-Lists for the original database and updates them whenever an incremental database is added. With the first request of the user to mine UFPs, the algorithm recursively constructs ICUP-Lists, extracts the UFPs, and stores the IUP-Lists and ICUP-Lists for the next stages. Then, it reports the results set, R, to the user and deletes them for memory efficiency. After that, the IUP-Lists are updated whenever a new database is added. If a mining request occurs from the user, the ICUP-Lists are updated, the UFPs are extracted, and the results are reported to the user.

3.3. Proposed method: List-based data structures for incremental mining of UFPs

In a dynamic environment, a database grows frequently meaning that new data are continually added. The review of the most important algorithms of uncertain frequent pattern mining in the previous section showed that none of them are proper for mining uncertain frequent patterns from incremental data. Indeed, an efficient method for working with

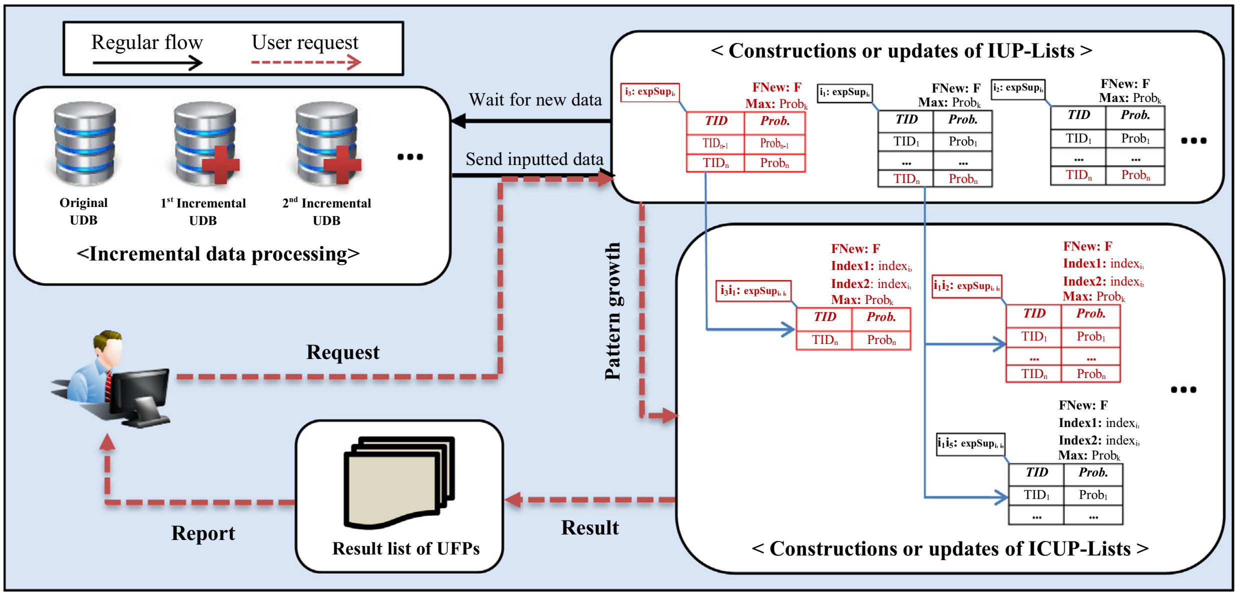


Fig. 1. Overall architecture of proposed ILUNA algorithm.

incremental environments should have the following capabilities: 1) All of the necessary operations to mine the UFPs should be done within a single database scan (read-only once); 2) When a new database is added, there is no need to rebuild the data structure to store database information from scratch (build-once). Hence, similar to the LUNA algorithm [27], the proposed ILUNA method uses a list structure that can satisfy the above requirements on incremental environments. In this method, a list-based structure similar to UP-List is used to store the database information and a list-based structure similar to CUP-List is recursively constructed to mine valid UFPs.

One solution to handle incremental data is to keep only the UP-Lists information for the next stages. That is, whenever the original database is entered, the UP-Lists information is stored and when a new database is added, the existing UP-Lists are updated and new UP-Lists are created only for new items. At the end, all the UP-Lists are sorted in support ascending order and the process of constructing CUP-Lists is done from the beginning. The implementation of this method and its evaluation results showed that this method is not very efficient compared to the LUNA algorithm. Therefore, a more effective method is suggested in this study.

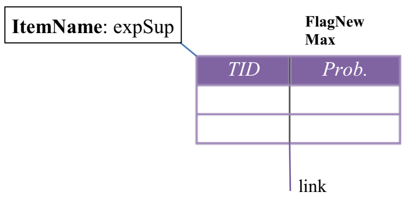
To handle incremental data, the proposed ILUNA method adopts an efficient method to update the UP-Lists and CUP-Lists which does not require repeating the mining process from scratch. Indeed, the proposed ILUNA benefits from knowledge obtained in previous stages. That is, after scanning the database and extracting the UFPs, the information of UP-Lists and CUP-Lists are maintained for the next stages. In the next stages, this information only will be updated if necessary. To update the lists, the current structures of the UP-List and CUP-List are not very efficient. Indeed, more features should be added to them. For this purpose, two new data structures called IUP-List and ICUP-List will be proposed. The IUP-List and ICUP-List are similar to the UP-List and CUP-List structures, respectively except that the IUP-List stores an additional parameter called FlagNew, and the ICUP-List stores three additional parameters called FlagNew, Index1, and Index2. Besides, each IUP-List or ICUP-List uses a link that refers to the set of ICUP-Lists generated from it.

Definition 5 ((***I**ncremental **U**ncertain **P**robability-**List** (IUP-List)*)). The proposed IUP-List is a data structure for maintaining the database information that is constructed for each distinct item through a database scan. Like the case of the UP-List, it includes a set of tuples containing TID information of the transactions with the current item and the corresponding existential probabilities. It stores the item name and expSup for the corresponding item and the maximum value of the existential probabilities. Moreover, it stores a check status flag called FlagNew which indicates the status of the IUP-List after the new database is added; whenever by entering a new database, a new IUP-List is created for a new item, its Flagnew becomes true. Also, each IUP-List has a link that refers to the set of ICUP-Lists generated from it. Fig. 2(a) shows a general architecture of IUP-List.

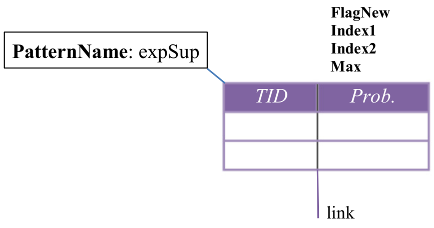
After introducing the IUP-List structure, the details of the ICUP-List structure will be suggested as follows.

Definition 6 ((***I**ncremental **C**onditional **U**ncertain **P**robability-**List** (ICUP-List)*)). The proposed ICUP-List is a data structure for mining UFPs that is constructed of combining a pair of IUP-Lists or ICUP-Lists. It is similar to the CUP-List consisting of a set of tuples with the TID information of the transactions containing the pattern and the corresponding existential probability information. It stores the name and expSup value of the pattern and the maximum existential probability value between its tuples. Moreover, it stores a check status flag; FlagNew. When a new ICUP-List is created, its FlagNew will be true. Also, it stores two indexes namely Index1 and Index2. Since each ICUP-List is created by merging two IUP-Lists or ICUP-Lists, the Index1 and Index2 show the indexes of the last processed tuples of the first and second lists, respectively. Each ICUP-List has a link that refers to the set of ICUP-Lists generated from it. Fig. 2(b) shows a general architecture of ICUP-List.

It should be noted that in the LUNA algorithm, there is no link from an/a UP-List or CUP-List to CUP-Lists constructed from it because after processing each UP-List or CUP-List, the set of CUP-Lists generated from it is immediately processed and the UFPs are recursively extracted from them so that there is no need for a link. However, in the proposed algorithm, this information is used in the next stages. Therefore, this link allows the information to be stored effectively so that the algorithm can easily access this information in the next stages.



(a) General architecture of IUP-List



(b) General architecture of ICUP-List

Fig. 2. The general architecture of IUP-List and ICUP-List.

3.3.1. The process of generating or updating of IUP-Lists

At first, the given original database is scanned once and transactions data are stored into IUP-Lists. Indeed, the proposed algorithm generates the IUP-Lists for all frequent and infrequent items with FlagNew = true and continually updates the maximum value of the probabilities of the IUP-Lists. After that, constructed IUP-Lists with FlagNew = true are sorted in support ascending order while infrequent items are not pruned. After completing the first mining operation, whenever a new database is added, the algorithm scans it once and processes transactions in the following sequence. If the IUP-List for an item is available, the corresponding IUP-List is updated; otherwise, a new IUP-List is created for the item with FlagNew = true and is stored at the beginning of the lists. The order of the IUP-Lists made in the previous stages remains constant and only newly created IUP-Lists with FlagNew = true are sorted in ascending order of frequency. After sorting new IUP-Lists, their FlagNews are set to false for the next stages.

3.3.2. The process of mining UFPs by generating or updating ICUP-Lists recursively

The mining process is similar to the LUNA algorithm in a recursive divide-and-conquer manner, the ICUP-Lists are constructed only from pairs of IUP-Lists or ICUP-Lists that are both frequent. In the mining process, the processing of lists starts from the beginning of the lists and any frequent IUP-List or ICUP-List will be combined with its next frequent lists to create new ICUP-Lists or update them. For the original database, the ICUP-Lists are generated with FlagNew = true by combining two frequent IUP-Lists or ICUP-Lists. When the incremental database is added, if an ICUP-List is not available for a frequent pattern, it is created with FlagNew = true; otherwise, it is updated if needed.

To update the ICUP-Lists, the proposed algorithm adopts an effective method named PNT technique that eliminates additional calculations. This technique will be introduced in Definition 7. Before that, to better understand how to update the ICUP-Lists by the proposed algorithm, the following example is provided.

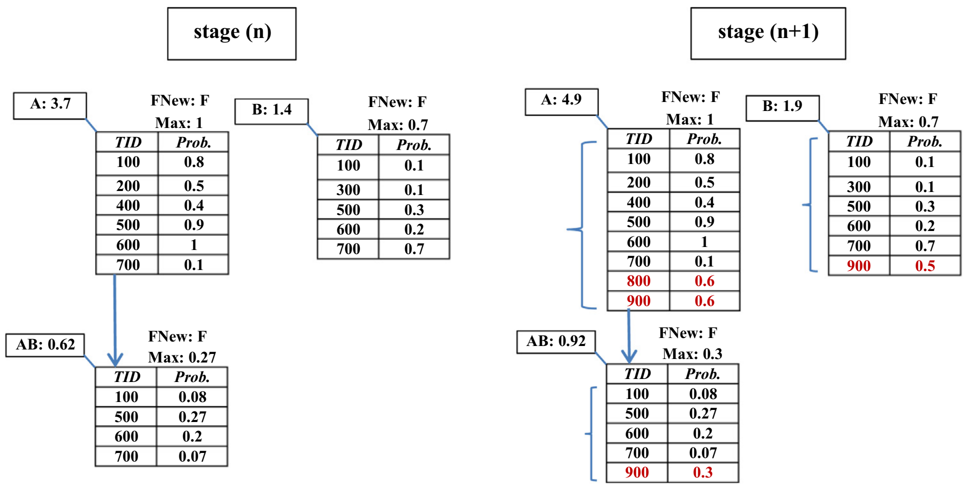
Example1. Consider in stage (n) of Fig. 3(a), the ICUP-List for AB has been made by combining A's IUP-List and B's IUP-List. With the addition of the new database in stage (n + 1), two tuples to the A's IUP-List and one tuple to the B's IUP-List are added. Now to update the AB's ICUP-List, a naïve manner is to check all the tuples set of A's IUP-List and B's IUP-List from scratch. In this manner, for each common tuple that is found, the algorithm must search the tuples set of the AB's ICUP-List. If this common tuple is not in the AB's ICUP-List, then the algorithm adds it to the AB's ICUP-List as a new tuple, as can be seen in the stage (n + 1) of Fig. 3(a). This method is a very costly solution because it is very time-consuming to re-process all the tuples set of lists that have been processed in the previous stages. This is while the AB's ICUP-List can be updated only with the processing of new tuples of A's IUP-List and B's IUP-List. To increase efficiency, the proposed algorithm considers Index1 and Index2 for the ICUP-Lists. Index1 and Index2 of AB's ICUP-List show what tuples from A's IUP-List and B's IUP-List have been processed for constructing the AB's ICUP-List in previous stages. As mentioned earlier in definition 6, Index1 = 5 and Index2 = 4 of AB's ICUP-List in stage (n) from Fig. 3(b) show the indexes of the last processed tuples of A's IUP-List and B's IUP-List, respectively. Accordingly, new tuples can be easily found by Index1 and Index2 and for updating the AB's ICUP-List in stage (n + 1), it is only necessary to process tuples of A's IUP-List with the index higher than 5 (i.e., TID:800 and TID:900) and tuples of B's IUP-List with the index higher than 4 (i.e., TID:900). The result (i.e., TID: 900) is added to AB's ICUP-List as a new tuple and finally, Index1 and Index2 of AB's ICUP-List are updated for the next stages (i.e., Index1 = 7 and Index2 = 5).

In some cases, an ICUP-List has been created in the previous stages but with the entry of a new database, it does not need to be updated. The proposed algorithm uses an intelligent approach to quickly detect these lists so that there is no need to process them. To better understand how the proposed algorithm works in this case, the following example is presented.

Example 2. Consider in stage (n) of Fig. 4, the ICUP-List for DC has been made by combining D's IUP-List and C's IUP-List. With the addition of the new database in stage (n + 1), three tuples to the C's IUP-List are added. To update the DC's ICUP-List, a naïve manner is to check all the tuples set of D's IUP-List and C's IUP-List from scratch so that the algorithm understands that these lists do not have a new common tuple and therefore the DC's ICUP-List does not need to be updated. But this method is also time-consuming. Instead, before processing the DC's ICUP-List, the proposed algorithm realizes that DC's ICUP-List does not need to be updated by checking its Index1 and Index2. Consider the fact that when the D's IUP-List or C's IUP-List or both are not changed, the DC's ICUP-List also remains unchanged so that it does not need to be updated. Thereby, if Index1 or Index2 of DC's ICUP-List or both are equal to the last tuples indexes of D's IUP-List and C's IUP-List, respectively, it means that D's IUP-List or C's IUP-List or both have not been updated. Consequently, DC's ICUP-List does not need to be updated and only the values of Index1 and Index2 of DC's ICUP-List are updated for the next stages. As can be seen in stage (n + 1) of Fig. 4, Index1 = 3 of DC's ICUP-List is equal to the last tuple index of D's IUP-List because the D's IUP-List has not been changed with the entry of the new database. Indeed, if the proposed algorithm does not use Index1 and Index2, it has to process all tuples set of the D's IUP-List, C's IUP-List, and DC's ICUP-List, while there is no need to do so.

As a result, example1 and example2 clearly show that the use of Index1 and Index2 can dramatically increase the efficiency of the proposed method.

According to the explanations provided, when an incremental database is added, consider U_1 as the current frequent list that is being processed and U_2 as one of its subsequent frequent lists. When U_1 and U_2 are combined to generate or update ICUP-Lists, U_1 is faced with three different cases:



Stage (n), A:

TID	Prob.
100	0.8
200	0.5
400	0.4
500	0.9
600	1
700	0.1

Stage (n), B:

TID	Prob.
100	0.1
300	0.1
500	0.3
600	0.2
700	0.7

Stage (n), AB:

TID	Prob.
100	0.08
500	0.27
600	0.2
700	0.07

Stage (n+1), A:

TID	Prob.
100	0.8
200	0.5
400	0.4
500	0.9
600	1
700	0.1
800	0.6
900	0.6

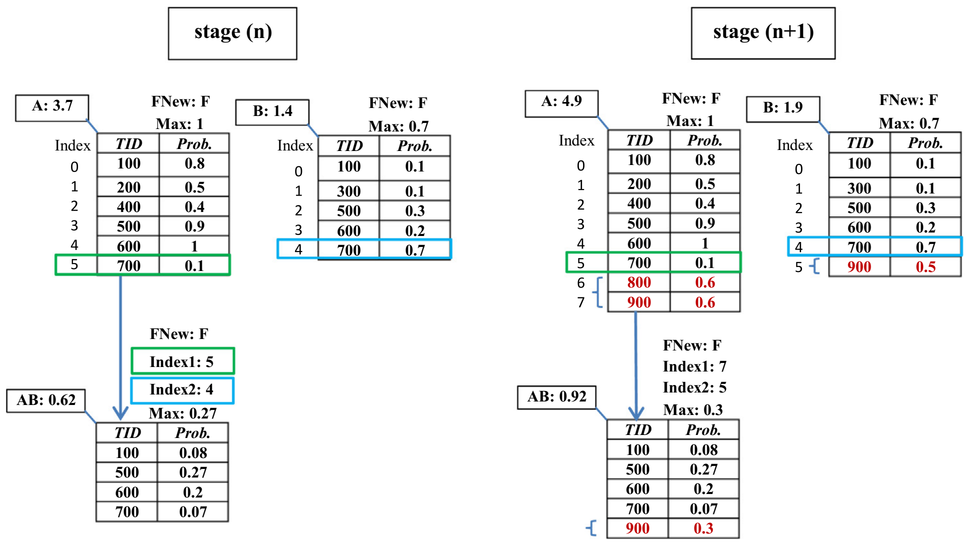
Stage (n+1), B:

TID	Prob.
100	0.1
300	0.1
500	0.3
600	0.2
700	0.7
900	0.5

Stage (n+1), AB:

TID	Prob.
100	0.08
500	0.27
600	0.2
700	0.07
900	0.3

(a) Naïve manner for updating ICUP-Lists



Stage (n), A:

Index	TID	Prob.
0	100	0.8
1	200	0.5
2	400	0.4
3	500	0.9
4	600	1
5	700	0.1

Stage (n), B:

Index	TID	Prob.
0	100	0.1
1	300	0.1
2	500	0.3
3	600	0.2
4	700	0.7

Stage (n), AB:

TID	Prob.
100	0.08
500	0.27
600	0.2
700	0.07

Stage (n+1), A:

Index	TID	Prob.
0	100	0.8
1	200	0.5
2	400	0.4
3	500	0.9
4	600	1
5	700	0.1
6	800	0.6
7	900	0.6

Stage (n+1), B:

Index	TID	Prob.
0	100	0.1
1	300	0.1
2	500	0.3
3	600	0.2
4	700	0.7
5	900	0.5

Stage (n+1), AB:

TID	Prob.
100	0.08
500	0.27
600	0.2
700	0.07
900	0.3

(b) Proposed manner for updating ICUP-Lists

Fig. 3. Comparing the naïve and proposed methods for updating ICUP-Lists.

Case 1. There is no ICUP-List for U_1U_2. In this case, U_1 and U_2 are combined to create a new ICUP-List for U_1U_2. Therefore, the algorithm should compare all tuples set of U_1 with all tuples set of U_2 to find common tuples to generate a new ICUP-List for U_1U_2.

Case 2. The ICUP-List for U_1U_2 has been made in previous stages and its Index1 and Index2 are not equal to the last tuples indexes of U_1 and U_2, respectively. This means that U_1 and U_2 have been updated with the addition of a new database and new tuples have been added to their tuples set. In this case, U_1U_2's ICUP-List needs to be updated. Therefore, it is enough that the algorithm processes only new tuples of U_1 and U_2 for combining the common tuples and adding them to the existing tuples of U_1U_2. As discussed in Example 1, the PNT technique introduced in Definition 7 is used to process new tuples.

Definition 7. ((***P****rocessing* ***N****ew* ***T****uples (PNT technique)*)). In a situation where the ICUP-List for U_1U_2 needs to be updated, the algorithm can easily process only new tuples of U_1 and U_2 using Index1 and Index2 of U_1U_2's ICUP-List. To do this, the algorithm reads Index1 and Index2 of U_1U_2's ICUP-List and finds Index1 on U_1 and Index2 on U_2, and processes their subsequent tuples as new tuples.

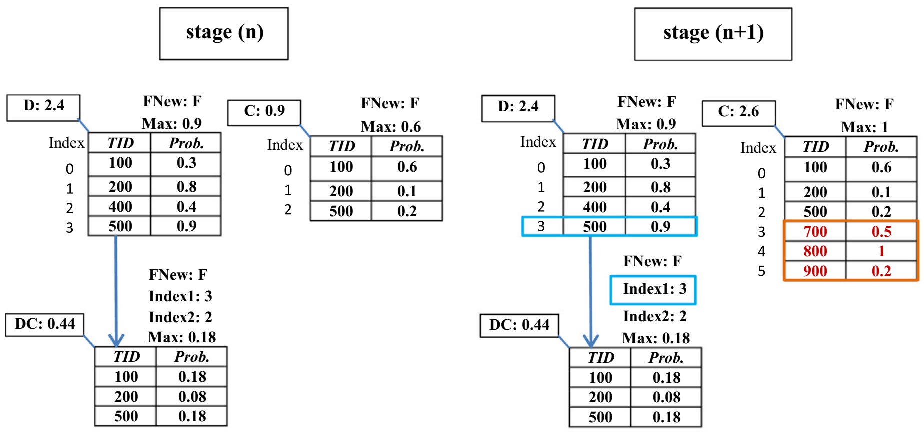


Fig. 4. Checking the cases that the ICUP-Lists do not need to be updated.

Case 3. The ICUP-List for U_1U_2 has been made in previous stages and Index1 or Index2 of U_1U_2's ICUP-List or both are equal to the last tuples indexes of U_1 and U_2, respectively. This means that U_1 or U_2 or both have not been updated. Therefore, U_1U_2 does not need to be updated and only the values of Index1 and Index2 of U_1U_2's ICUP-List are updated for the next stages (As discussed in Example 2).

After processing U_1, when all possible ICUP-Lists are constructed or updated from U_1, the proposed algorithm sorts new ICUP-Lists with FlagNew = true in support ascending order. Then, the algorithm recursively constructs or updates the ICUP-Lists from them in a depth-first-search manner for mining longer patterns. The LUNA algorithm does not sort the CUP-Lists, but the proposed algorithm sorts the ICUP-Lists in the same way as the IUP-Lists, which causes a smaller number of ICUP-Lists to be generated in the mining process. As a result, it helps to improve the performance of the algorithm.

It should be noted that in [27], to improve the performance of the LUNA algorithm, some techniques are used to reduce the search space and redundant mining operations such as pre pruning factor (ppf) technique and improve the mining speed without additional use of memory and several other techniques. For more details, please refer to [27]. The proposed algorithm also employs all of these techniques.

The following examples are provided to better understand how the proposed algorithm works when adding incremental databases.

Example 3. Consider the original UDB_0 in Table 2 when the user-defined minimum support threshold is set at 0.1 and therefore the minSup = 0.3 (=0.1 $\times$ 3). In the original database (UDB_0), when reading the first transaction {a:0.8, b:0.1, c:0.2}, three IUP-Lists are created for A, B, and C items with FlagNew = true. Each IUP-List holds the name, expSup, and maximum existential probability of the corresponding item and a set of tuples including TID of the transactions with the current item and the corresponding existential probabilities. In the same manner, transactions TID:200 and TID:300 are processed (Fig. 5(a)). After the scan is completed, the IUP-Lists with FlagNew = true are arranged in support ascending order and their FlagNew are set to false. As shown in Fig. 5(b), the order of the IUP-Lists is A-B-C. By requesting mining operation, the algorithm first processes frequent item A and checks items B and C to combine with it. Because the item B is infrequent, the ICUP-List for pattern AB is not generated. However, A is combined with the frequent item C and the ICUP-List is created for pattern AC. For its construction, the algorithm compares all tuples of A with all tuples of C to find common tuples and combine them for generating a new ICUP-List for AC. The index of the last processed tuple of A's IUP-List is 1 and the index of the last processed tuple of C's IUP-List is 2; therefore, the AC's ICUP-List stores the Index1 = 1 and Index2 = 2. At this stage, processing A is over. The next item to process is B, but the algorithm does not process it because it is infrequent. The next item is C; however, the algorithm terminates because there is no subsequent IUP-List for combining. The results of mining frequent patterns from the original database are A and C.

The following example illustrates how the algorithm works after adding a new database.

Example 4. Consider inserting UDB_1 of Table 2 while minSup is calculated as minSup = 0.5(0.1$\times$5). The algorithm scans the UDB_1 once and generates IUP-Lists for D and E with FlagNew = true and IUP-Lists for A and B are updated (Fig. 6(a)). The order of the IUP-Lists is not changed for the items A, B, and C that have been made in the previous stage. Only the IUP-Lists for D and E are sorted as shown in Fig. 6(b). Once the UDB_1 scanning process is complete, when the user requests to mine patterns, item D is not processed because it is infrequent and the proposed ILUNA starts by processing item E. The IUP-List for E combines with all its subsequent frequent IUP-Lists, and therefore ICUP-Lists for EA and EB are generated while ICUP-List for EC is not made because C is infrequent (Fig. 6(b)). Then, the algorithm sorts the ICUP-Lists for EA and EB in support ascending order and sets their FlagNew to false. Thereafter, it recursively generates an ICUP-List for EAB by processing the

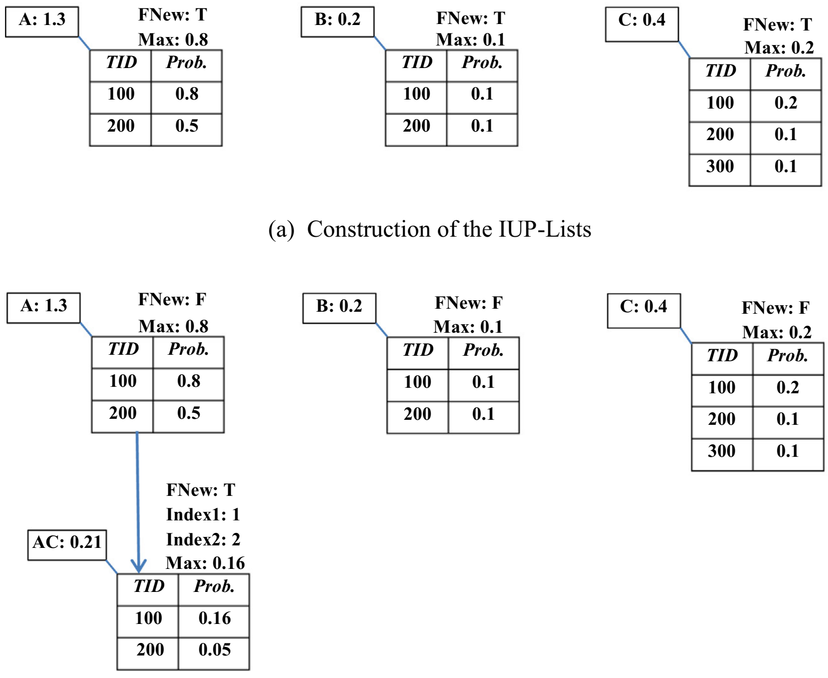
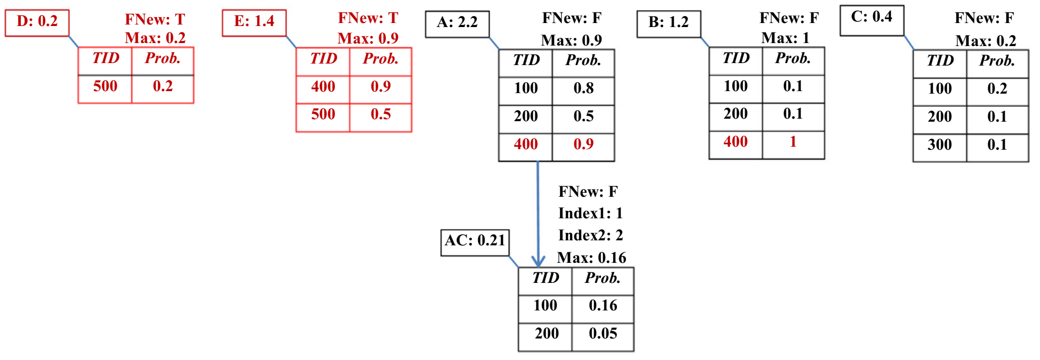


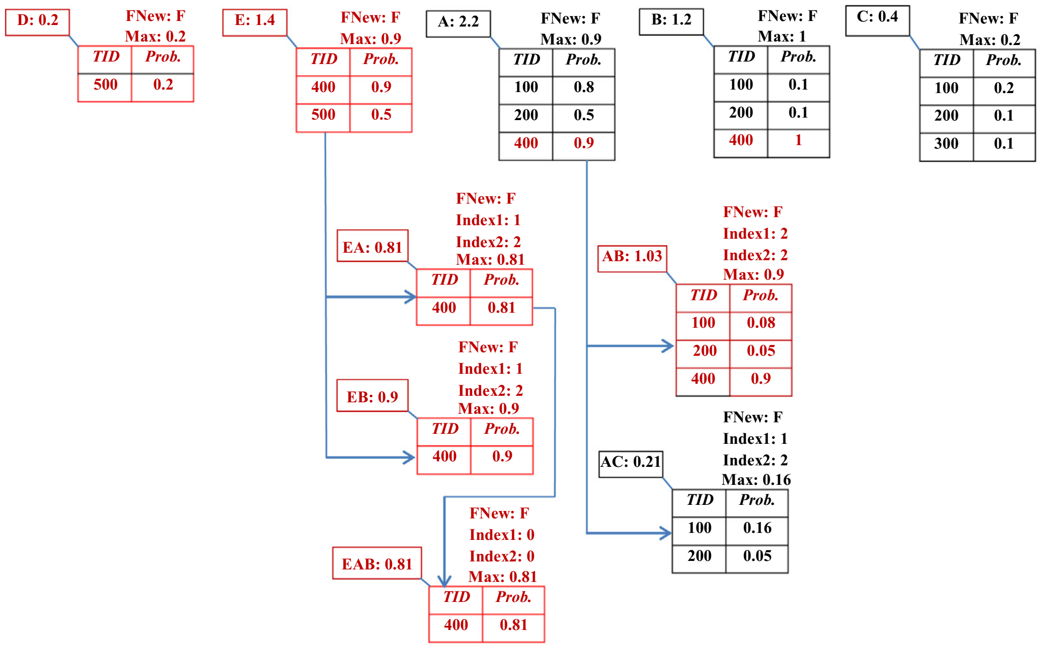
Fig. 5. The mining process of UFPs for original database with minSup = 0.3.

EA's ICUP-List. After processing item E, the algorithm processes item A which its subsequent items are B and C. The B has become frequent with the addition of the new database, therefore the A and B are combined. The ICUP-List for AB has not been created in the previous stage. Therefore, the proposed ILUNA generates it with FlagNew = true. The next item C is infrequent; thus, the algorithm does not process AC's ICUP-List. The algorithm has to sort the new ICUP-Lists with FlagNew = true made from A; however, there is only one new AB's ICUP-List so the order of the ICUP-Lists is not changed. Then, the algorithm recursively processes the AB's ICUP-List, but ICUP-List for ABC is not created because the AC pattern is infrequent. After processing item A, the algorithm processes item B. The B's IUP-List has not been processed in the previous stage due to being infrequent. Therefore, it must be combined with all its subsequent frequent items, but the C is not frequent. Hence, the ICUP-List for BC is not made. At the end, the extracted UFPs are E, A, B, EA, EB, AB, and EAB. How to add the UDB_2 is discussed below.

Example 5. Consider inserting UDB_2 of Table 2 while minSup is calculated as minSup = $0.7(0.1\times7)$. By scanning UDB_2 once, the IUP-Lists for D, A, B, and C are updated. Since no new IUP-List is created, there is no need to sort the IUP-Lists (Fig. 7(a)). By adding the new database UDB_2, item D becomes frequent. Therefore, the algorithm processes it and constructs ICUP-Lists for DE, DA, DB, and DC and sorts them. The order of the ICUP-Lists becomes DE-DB-DC-DA and their FlagNew are set to false. Their final order is shown in Fig. 7(b). The DE, DB, and DC patterns are infrequent; therefore, they cannot satisfy the conditions for generating ICUP-lists for longer patterns. Then, the algorithm processes the E's IUP-List. The E and A patterns are both frequent; therefore the algorithm first checks the EA's ICUP-List. Because Index1 of EA's ICUP-List is equal to the last tuple index of E's IUP-List (Index1 = 1), the EA's ICUP-List does not need to be updated and only the value of Index2 of EA's ICUP-List is updated from value 2 to value 4. Then, the algorithm checks the subsequent pattern B. The E and B patterns are both frequent so the algorithm checks the EB's ICUP-List. Like the EA, the EB's ICUP-List does not need to be updated and only the value of Index2 of EB's ICUP-List is updated from value 2 to value 3. To merge the E and subsequent pattern C, since the E's IUP-List and C's IUP-List have no common tuples; therefore, EC's ICUP-List is not created from them. The algorithm recursively checks the EAB's ICUP-List by processing EA's ICUP-List; it does not need to be updated because its Index1 and Index2 are both equal to the last tuples indexes of ICUP-Lists for EA and EB, respectively. The algorithm processes the A's IUP-List and subsequently checks the AB's ICUP-List. Because Index1 and Index2 of the AB's ICUP-List are different from the last tuples indexes of A's IUP-List and B's IUP-List, the AB's ICUP-List must be updated. Based on the PNT technique, comparing the new tuples of the A's IUP-List (i.e., TID: 600 and TID: 700) and B's IUP-List (i.e., TID: 600) is enough. The algorithm checks Index1 and Index2 of the AB's ICUP-List. Index1 and Index2 are 2; therefore the algorithm processes only subsequent tuples of index = 2 in IUP-Lists for A and B. The result is a new tuple (i.e., TID: 600) that is added to the AB's ICUP-List, and its



(a) Updating of the IUP-Lists after adding transactions UDB$_1$



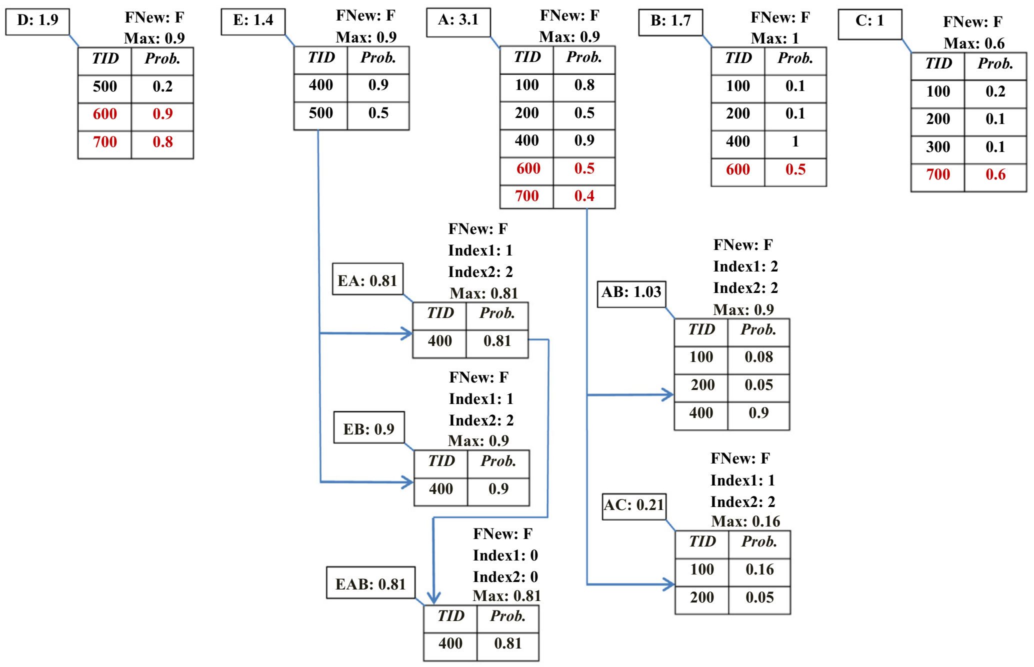
(b) Updating of the ICUP-Lists after adding transactions UDB$_1$

Fig. 6. The mining process of UFPs after adding UDB$_1$ with minSup = 0.5.

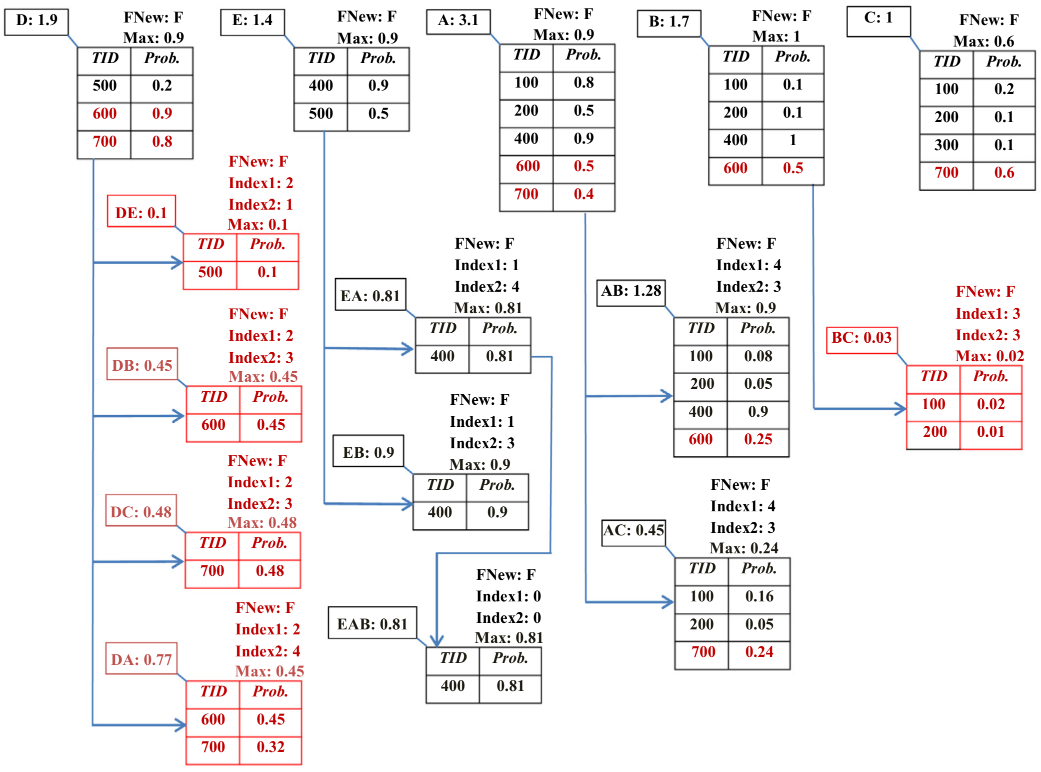
existential probability value is added to the expSup value of the AB's ICUP-List. Similarly, the AC's ICUP-List is updated and a new tuple with TID:700 is added to it. At the end, the algorithm processes item B. Although its next item C is frequent, the ICUP-List for BC has not been made in the previous stages. Because the C was frequent and B was infrequent when the original database was scanned and the B was frequent and the C was infrequent when the UDB$_1$ was added, so the ICUP-List for BC was not made in the previous stages. At this stage, both are frequent so the ICUP-List for BC is made by comparing all tuples of B's IUP-List and C's IUP-List. At the end of the mining operation, the resulting set is D, E, A, B, C, DA, EA, EB, AB, EAB.

3.3.3. ILUNA algorithm

In this section, the overall pseudo-code of the ILUNA algorithm has been depicted in Fig. 8. The algorithm first calls sub-procedure Construct_IUP-Lists for storing the database data in the IUP-Lists (Line 1). Indeed, it constructs the IUP-Lists for new items or updates the previously constructed IUP-Lists. For the original database, all the IUP-Lists are sorted in their support ascending order (Line 2). When a mining request is mad by the user (Line 3), the algorithm calls sub-procedure Mine_Patterns to extract UFPs and store them into the results set, R (Line 4). Whenever the mining works are finished, the algorithm reports the results set, R, to the user. After processing the original database, the Construct_IUP-Lists and



(a) Updating of the IUP-Lists after adding transactions UDB$_2$



(b) Updating of the ICUP-Lists after adding transactions UDB$_2$

Fig. 7. The mining process of UFPs after adding UDB$_2$ with minSup = 0.7.

```
Main_Procedure: ILUNA
Input: An original database, UDB₀
        A set of incremental databases, IncUDB={UDB₁, UDB₂, UDB₃, …}
        A user-defined minimum support threshold, minSup
Output: a set of UFP results, R
Variable: a set of IUP-Lists, IU
          a set of ICUP-Lists, IC
          a prefix itemset, pref
1.   Call Construct_IUP-Lists (UDB₀)
2.   Sort items in their support ascending order
3.   IF a mining request is made, then
4.       Call Mine_Patterns (IU, minSup)
5.   End IF
6.   For each incremental database, UDBᵢ, in IncUDB
7.     Call Construct_IUP-Lists (UDBᵢ)
8.     Sort new items with FlagNew=true in their support ascending order and add them to the top of the items
9.     IF a mining request is made, then
10.        Call Mine_Patterns (IU, minSup)
11.    End IF
12.  End For
```

Fig. 8. Main procedure of ILUNA algorithm.

Mine_Patterns procedures are repeatedly called for the incremental databases (Lines 6 to 12). The difference is that for incremental databases, the order of the IUP-Lists made in the previous stages remains unchanged and only the new IUP-Lists with FlagNew = true are sorted in support ascending order and added to the top of the IUP-Lists (Line 8). In this way, the results of mining UFPs can be obtained from given incremental databases. Fig. 9 shows the sub-procedures of the proposed algorithm.

The first sub-procedure, Construct_IUP-Lists, is called to construct new IUP-Lists or update existing IUP-Lists (Lines 1 to 8). The ILUNA algorithm first scans the original or incremental database (Line 1). By scanning each transaction, if there is no IUP-List for each item, the proposed ILUNA creates an IUP-List with FlagNew = true (Lines 2 and 3) and stores constructed IUP-List into a set of IUP-Lists for the next mining task (Line 4); otherwise (Line 5), if there is an IUP-List for the item, the algorithm updates existing IUP-List (Line 6).

The second sub-procedure, Mine_patterns, constructs ICUP-Lists from IUP-Lists or updates them to extract UFPs (lines 9 to 36). In general, for combining each IUP-List, u_k, with each of its subsequent IUP-List, u_l, the Mine_patterns procedure can be divided into two main parts: i) the ICUP-List from u_k and u_l has not been created in the previous stages, in which case the algorithm generates an ICUP-List for u_ku_l, ii) the ICUP-List from u_k and u_l has been created in the previous stages, in which case the algorithm updates existing u_ku_l's ICUP-List. The details of these steps are explained in the following. For each IUP-List, u_k (Line 9), if u_k be frequent (Line 10), the algorithm adds it into the results set as a valid UFP (Line 11). To construct or update the ICUP-Lists, the u_k is combined with its subsequent IUP-Lists. For this purpose, if a set of ICUP-Lists does not exist for u_k, the algorithm initializes a set of ICUP-Lists, IC, and a prefix itemset, pref (Line 12), and sets the item of the currently selected IUP-List to pref (Line 13). Then, for each IUP-List, $u_{l,}$ after the u_k (Line 14), if u_l is frequent (Line 15), the algorithm performs subsequent tasks for constructing an ICUP-List from u_k and u_l. The proposed ILUNA first checks the pruning technique (Line16) and if the u_l is not pruned then the algorithm checks if the ICUP-List for u_ku_l has not been built in the previous stages (Line 17), it generates an ICUP-List from u_k and u_l, c, with FlagNew = true (Line 18). Thereafter, it sets Index1 and Index2 values of c with indexes values of u_k's last processed tuple and u_l's last processed tuple, respectively (Line 19), and adds c into a set of ICUP-Lists for the next mining task (Line 20). Otherwise (Line 21), if the ICUP-List for u_ku_l, c, has already been made and its Index1 and Index2 are not equal to the indexes of u_k's last tuple and u_l's last tuple, respectively (Line 22), the algorithm updates the existing u_ku_l's ICUP-List according to PNT technique (Line 23). After that, the algorithm updates Index1 and Index2 values of c (line 25). After the u_k processing is complete, if the number of constructed ICUP-Lists from u_k is higher than 1, the algorithm sorts new ICUP-Lists with FlagNew = true in support ascending order, adds them to the top of the ICUP-Lists, and calls sub-procedure ILUNA_growth to extract longer patterns (Lines 30 to 33).

The third sub-procedure, ILUNA_growth extracts UFPs with 3 or longer lengths recursively (Lines 37 to 63). To do this, it works similar to the part between lines 9 to 36 in the sub-procedure Mine_Patterns except that the algorithm continually updates the prefix information for each call of the sub-procedure (Line 41) and recursively constructs or updates ICUP-Lists from previous ICUP-Lists until there is no more ICUP-List to be combined.

4. Analysis of time complexity

In this section, theoretically, the time complexity of the proposed ILUNA algorithm is compared to LUNA [27] and Incremental-CUF-growth [25]. Besides, the empirical performance is investigated via various real and synthetic datasets in the next section. Thus, the superiority of the proposed algorithm is proved in both theoretical and empirical aspects.

```
Sub_Procedure: Construct_IUP-Lists ( UDB$_0$ or IncUDB)
1.  For items of each $T_j$ in UDB$_0$ or IncUDB
2.      If IUP-List, $u_k$ does not exist for the item
3.          Generate an $u_k$ for the item with FlagNew=true and set its name, max, expSup values
4.          IU ← IU ∪ $u_k$
5.      Else
6.          update the tuples, expSup, and max information of the corresponding $u_k$
7.      End if
8.  End for

Sub_procedure: Mine_Patterns (a set of IUP-Lists, IU, minSup)
9.  For each $u_k$ in IU
10.     If ($u_k$.expSup ≥ minSup)
11.         R ← R ∪ $u_k$.pattern
12.         If a set of ICUP-Lists does not exist for the $u_k$, then initialize a set of ICUP-Lists, IC and pref
13.         Insert $u_k$.item into pref
14.         For each $u_l$ in IU // k<l
15.             If ($u_l$.expSup ≥ minSup)
16.                 If (($u_k$.expSup * max of $u_l$) ≥ minSup)
17.                     If ICUP-List from $u_k$ and $u_l$ does not exist
18.                         Generate an ICUP-List, c, from $u_k$ and $u_l$ with FlagNew=true
19.                         Set c.Index1 = index of $u_k$'s last processed tuple and c.Index2 = index of $u_l$'s last processed tuple
20.                         IC ← IC ∪ c
21.                     Else
22.                         If (c.Index1 != index of $u_k$'s last tuple and c.Index2 != index of $u_l$'s last tuple)
23.                             Update existing $u_k u_l$'s ICUP-List according to PNT technique
24.                         End if
25.                         Set c.Index1 = index of $u_k$'s last processed tuple and c.Index2 = index of $u_l$'s last processed tuple
26.                     End if
27.                 End if
28.             End if
29.         End for
30.         If (# of elements in IC>1)
31.             Sort new ICUP-Lists with FlagNew=true in their support ascending order and add them to the top of the ICUP-Lists
                and set their FlagNew to false
32.             Call ILUNA_growth (IC, pref, minSup)
33.         End if
34.     End if
35. End for
36. Return R

Sub_procedure: ILUNA_growth (a set of ICUP-Lists, IC, a prefix itemset, pref, minSup)
37. For each $c_k$ in IC
38.     If ($c_k$.expSup ≥ minSup)
39.         R ← R ∪ $c_k$.pattern
40.         If a set of ICUP-Lists does not exist for $c_k$, then initialize a set of ICUP-Lists, IC'
41.         Initialize a prefix itemset for the next stage, pref', with pref and $c_k$
42.         For each $c_l$ in IC // k<l
43.             If ($c_l$.expSup ≥ minSup)
44.                 If (($c_k$.expSup * max of $c_l$) ≥ minSup)
45.                     If ICUP-List from $c_k$ and $c_l$ does not exist
46.                         Generate an ICUP-List, c', from $c_k$ and $c_l$ with FlagNew=true
47.                         Set c'.Index1= index of $c_k$'s last processed tuple and c'.Index2 = index of $c_l$'s last processed tuple
48.                         IC' ← IC' ∪ c'
49.                     Else
50.                         If ( c'.Index1!= index of $c_k$'s last tuple and c'.Index2 != index of $c_l$'s last tuple)
51.                             Update existing $c_k c_l$'s ICUP-List according to PNT technique
52.                         End if
53.                         Set c'.Index1= index of $c_k$'s last processed tuple and c'.Index2 = index of $c_l$'s last processed tuple
54.                     End if
55.                 End if
56.             End if
57.         End for
58.         If (# of elements in IC'>1)
59.             Sort new ICUP-Lists with FlagNew=true in their support ascending order and add them to the top of the ICUP-Lists
                and set their FlagNew to false
60.             Call ILUNA_growth (IC', pref', minSup)
61.         End if
62.     End if
63. End for
```

Fig. 9. Sub procedures of ILUNA algorithm.

4.1. Analysis of time complexity of ILUNA vs LUNA

In this subsection, the time complexity of proposed ILUNA and LUNA algorithms is calculated when an incremental mining process is performed. Table 3 shows symbol definitions for the time complexity of ILUNA and LUNA algorithms. Consider UDB is the whole database and UDB = UDB_0 + UDB_i where UDB_0 and UDB_i are the original and incremental uncertain databases, respectively.

The main processes of the LUNA and ILUNA algorithms can be divided into two steps as follows (i) the UP-Lists/IUP-Lists construction and (ii) the mining UPFs by constructing CUP-Lists/ICUP-Lists. The LUNA algorithm scans the UDB twice to construct UP-Lists. Then, it mines the UFPs by combining UP-Lists or CUP-lists and constructing CUP-Lists for longer patterns. Therefore, the time required to extract UFPs by the LUNA algorithm can be calculated as follows.

T_{LUNA} **=** Time for reading the whole UDB + Time for reading the whole UDB + Time for constructing UP-Lists from scratch for whole UDB **+** Time for constructing CUP-Lists from scratch for whole UDB

$$T_{LUNA} = T_{UDB} + T_{UDB} + T_{up} + T_{cup} = 2\,T_{UDB} + T_{up} + T_{cup} \tag{4}$$

The ILUNA algorithm constructs or updates IUP-Lists by only one UDB_i scan. Then, it extracts the UFPs by constructing or updating ICUP-Lists for longer patterns. Therefore, the time required to extract UFPs by the ILUNA algorithm can be calculated as follows.

T_{ILUNA} **=** Time for reading the original UDB_0 or incremental UDB_i + Time for constructing or updating IUP-Lists for UDB_i **+**-Time for constructing or updating ICUP-Lists for UDB_i

$$T_{ILUNA} = T_{UDB_i} + T_{u\text{ - }up} + T_{u\text{ - }cup} \tag{5}$$

As can be seen from Equations (4) and (5), the LUNA algorithm requires two database scans to extract patterns but the ILUNA algorithm extracts patterns with only one database scan. When an incremental database (UDB_i) is added, the LUNA algorithm should scan the whole database (UDB_0 + UDB_i) twice but the proposed algorithm only scans the new database (UDB_i) once and obviously T_{UDB_i} 2 T_{UDB}. The LUNA algorithm generates all UP-Lists from scratch for UDB, while the proposed ILUNA algorithm updates existing IUP-Lists for UDB_i; therefore, it is clear that $T_{u\text{-}up}$ is $\leq T_{up}$. Also, the LUNA algorithm repeats the whole process of constructing CUP-Lists from scratch for UDB. However, the proposed ILUNA algorithm only updates the ICUP-Lists for UDB_i; therefore, $T_{u\text{-}cup} \leq T_{cup}$, which results in $T_{ILUNA} \leq T_{LUNA}$.

4.2. Analysis of time complexity of ILUNA vs Incremental-CUF-growth

In this subsection, the time complexity of the proposed ILUNA and Incremental-CUF-growth algorithms is examined. Recall that with receiving each of the new databases, the ILUNA scans only the new database. However, the Incremental-CUF-growth algorithm not only scans the new database but also needs an additional scan of the entire database (original + new) to filter the false positives. Table 4 presents symbol definitions for the time complexity of ILUNA and Incremental-CUF-growth algorithms.

The worst case scenario of the proposed algorithm is when each transaction of UDB_i contains all distinct items and a given minSup threshold is 0. Since the largest number of list combinations occurs in this case, all IUP-Lists and ICUP-Lists are updated by reading the new database. The ILUNA algorithm scans the new database, sorts new 1-length UFPs, and extracts UFPs by updating the lists. Therefore, the time required to extract UFPs by the ILUNA algorithm is calculated as T_{ILUNA} =-T_{UDBi} + T_{P1} + T_L. Suppose the processing time of each item is 1, then T_{UDBi} is calculated as $O(n_t' \times n_i)$. Since in the worst case, each transaction of UDB_i contains all distinct items, n_i is equal to $n_{i(D)}$; therefore $T_{UDBi} = O(n_t' \times n_{i(D)})$. After scanning the new database, the ILUNA sorts new 1-length UFPs in support ascending order by quick sort technique. Therefore, T_{P1} is calculated as $O\left(n_i' Log^{n_i'}\right)$ that n_i' (n_i' $n_{i(D)}$) is the number of new items in UDB_i. Since there is no pruned pattern in the worst case, the ILUNA extracts all of the UFPs possible from P_1 by combining a pair of IUP-Lists or ICUP-Lists recursively until there is no more list to be combined. If $|P_k|$ be the number of elements in P_k in ILUNA ($1 \leq k \leq n_{i(D)}$)), then, $|P_k|$ can be calculated as follows: $|P_1|={}_{n_{i(D)}}C_1$, $|P_2|={}_{n_{i(D)}}C_2$, . . ., $|P_k|={}_{n_{i(D)}}C_k$, . . ., $|P_{n_{i(D)}}|={}_{n_{i(D)}}C_{n_{i(D)}}$. Therefore, the number of lists constructed from P_1 is equal to the number of elements in the power set of P_1 subtracting 1 (corresponding to an empty set) and is calculated $|P_1|$ + $|P_2|$ +

Table 3
Symbol definitions for the time complexity of ILUNA vs LUNA.

Symbol	Meaning
T_{UDB}	Time for reading the whole UDB, including the original UDB_0 and incremental UDB_i
T_{UDBi}	Time for reading the original UDB_0 or incremental UDB_i
T_{up}	Time for constructing UP-Lists from scratch for UDB
$T_{u\text{-}up}$	Time for updating IUP-Lists for UDB_i
T_{cup}	Time to mine UFPs by constructing CUP-Lists from scratch for UDB
$T_{u\text{-}cup}$	Time to mine UFPs by updating ICUP-Lists for UDB_i
T_{LUNA}	Time required to extract UFPs by the LUNA
T_{ILUNA}	Time required to extract UFPs by the proposed ILUNA

Table 4
Symbol definitions for the time complexity of ILUNA vs Incremental-CUF-growth.

Symbol	Meaning
n_t	The number of transactions of original UDB_0
n_t'	The number of transactions of incremental UDB_i
n_i	The average number of items in each transaction of UDB_0 or UDB_i
n_i'	The number of new items in UDB_i
$n_{i(D)}$	The number of distinct items in UDB
n_{FP}	The number of generated potential UFPs by Incremental-CUF-growth
N	The number of all items in UDB
P_1	A set of 1-length UFPs in UDB such that expSups of them $\geq$ minSup
P_K	A set of k-length UFPs in UDB such that expSups of them $\geq$ minSup
${}_kC_x$	Combinations of k elements in groups of $\times$ elements
T_{UDB}	Time cost for reading the whole UDB
T_{UDBi}	Time cost for reading the original UDB_0 or incremental UDB_i
T_{P1}	Time cost for sorting a set of 1-length UFPs, P_1
T_L	Time cost for updating IUP-Lists and ICUP-Lists for UDB_i
T_{recons}	Time cost for reconstructing original Incremental-CUF-tree
T_{mine}	Time cost for constructing all of the possible conditional trees from Incremental-CUF-tree
$T(C_i)$	Time cost for constructing a conditional tree for potential UFP, i
T_{filter}	Time cost for filtering all of the false positives from the result set of Incremental-CUF-growth
$T_{Inc\text{-}CUF}$	Time cost to extract UFPs by the Incremental-CUF-growth
T_{ILUNA}	Time cost to extract UFPs by the proposed ILUNA

$\ldots + |P_k| + \ldots + |P_{n_{i(D)}}| = n_{i(D)}C_1 + n_{i(D)}C_2 + \ldots + n_{i(D)}C_k + \ldots + n_{i(D)}\ C_{n_{i(D)}} = 2^{n_{i(D)}} - 1$. Hence, $T_L = O(2^{n_{i(D)}} - 1)$ and consequently time required to extract UFPs by the proposed ILUNA is calculated as follows.

$$T_{ILUNA} = T_{UDBi} + T_{P1} + T_L = O\left(\left(n_t' \times n_{i(D)}\right) + \left(n_i' \times Log^{n_i}\right) + \left(2^{n_{i(D)}}\right)\right) \quad (6)$$

Based on Equation (6), the time complexity of ILUNA to extract UFPs in dynamic environments is $O(2^{n_{i(D)}})$ by considering only the highest order term.

Meanwhile, the time complexity of Incremental-CUF-growth is calculated as follows. The worst case of the Incremental-CUF-growth occurs when, firstly, it reconstructs an original tree that has no common branches and all its branches are unsorted paths; secondly, the global tree reconstructed is also a tree that has no common branches, and thirdly, the minSup threshold is 0 and as a result, the algorithm produces all potential UFPs. The Incremental-CUF-growth algorithm scans the new database, sorts the set of 1-length UFPs, reconstructs the original tree, mines potential UFPs by constructing conditional trees from scratch, and finally filters false positives by scanning the entire database (original + new). Therefore, the time required to extract UFPs by the Incremental-CUF-growth algorithm is calculated as $T_{Inc\text{-}CUF} = T_{UDBi} + T_{P1} + T_{recons} + T_{mine} + T_{filter}$. The Incremental-CUF-growth algorithm sorts the 1-length UFPs in descending order of their total expected support. Since, in the worst case for incremental algorithms, there is no pruned pattern, the number of 1-length UFPs is $n_{i(D)}$. Therefore, T_{UDBi} and T_{P1} are calculated as $O(n_t' \times n_i)$ and $O(n_{i(D)}\ Log^{n_{i(D)}})$, respectively. T_{recons} of Incremental-CUF-growth is calculated as follows. Since the original tree, G_0, has no common branches; therefore, it has n_t branches and each branch has n_i $(1 \leq n_i \leq n_{i(D)})$ nodes. Hence, the T_{recons} to reconstruct original tree branches by merge sort approach is calculated as $O(n_t \times n_i\ Log^{n_i})$. As shown in Table 4, T_{mine} is the time cost for constructing all of the possible conditional trees from the global tree, C_i $(1 \leq i \leq n_{i(D)} - 1)$. Indeed, when a new database is scanned, the algorithm reconstructs only the original tree and performs the process of constructing conditional trees and extracting patterns from scratch. Therefore, $T_{mine} = T(C_1) + T(C_2) + \ldots + T(C_{n_{i(D)-1}})$, where the time required to build the global tree G, T(G), is excluded from T_{mine} because it has been already constructed in the previous stage. $T(C_i)$ is calculated as follows. To construct a conditional tree from a global Incremental-CUF-tree, the algorithm scans the tree two times, sorts items, and constructs smaller conditional trees from the current conditional tree, recursively. In the worst case, the number of nodes of the global tree is N because any node in the tree does not merge with the others. Also, the current conditional tree contains $(n_{i(D)} - i)$ distinct items. If $RT(C_i)$ is the time required for conducting recursive processes for C_i and $\times$ be the number of constructed conditional trees from C_i, then, $RT(C_i) = T(C_{i,1}) + T(C_{i,2}) + \ldots + T(C_{i,x})$. Indeed, $T(C_{i,m})$ $(1 \leq m \leq x)$ works similar to the $T(C_i)$ except that it scans a smaller number of nodes and smaller items are sorted in the recursive processes. Therefore, T_{mine} is calculated as $O(\sum_{k=1}^{n_{i(D)}-1}(2 \times N + (n_{i(D)} - k)Log_2^{(n_{i(D)}-k)} + RT(C_k)))$. To filter the false positives, the algorithm must calculate the exact expSup of all potential UFPs by scanning the entire database. Since there is no pruned pattern in the worst case, the number of potential UFPs extracted by the algorithm is $n_{FP} = (2^{n_{i(D)}}-1)$ and the time required to scan the entire database is $T_{UDB} = O((n_t + n_t')\times n_i) = O(N)$. Hence, T_{filter} becomes $O((2^{n_{i(D)}}-1) \times N)$. Consequently, the time required to extract UFPs by the Incremental-CUF-growth is calculated as follows.

$$\begin{aligned}
\mathrm{T}_{\mathrm{Inc-CUF}} &= \mathrm{T}_{\mathrm{UDBi}} + \mathrm{T}_{\mathrm{P1}} + \mathrm{T}_{\mathrm{recons}} + \mathrm{T}_{\mathrm{mine}} + \mathrm{T}_{\mathrm{filter}} \\
&= O((\mathrm{n'}_{\mathrm{t}} \times \mathrm{n}_{\mathrm{i}}) + \left(\mathrm{n}_{\mathrm{i(D)}}\ Log_2^{\mathrm{n}_{\mathrm{i(D)}}}\right) + (\mathrm{n}_{\mathrm{t}} \times \mathrm{n}_{\mathrm{i}}\ Log^{\mathrm{n}_{\mathrm{i}}}) + (\sum_{k=1}^{n_{i(D)-1}} (2 \times N + \left(n_{i(D)} - k\right) Log_2^{\left(n_{i(D)}-k\right)} + RT(C_k))) + ((2^{n_{\mathrm{i(D)}}} \\
&\quad - 1) \times \mathrm{N})) \\
&= O((\mathrm{n'}_{\mathrm{t}} \times \mathrm{n}_{\mathrm{i}}) + (\mathrm{n}_{\mathrm{i(D)}} Log_2^{\mathrm{n}_{\mathrm{i(D)}}}) + (\mathrm{n}_{\mathrm{t}} \times \mathrm{n}_{\mathrm{i}}\ Log^{\mathrm{n}_{\mathrm{i}}}) + (2 \times N \times \left(n_{i(D)} - 1\right)) + (\sum_{k=1}^{n_{i(D)-1}} (\left(n_{i(D)} - k\right) Log_2^{\left(n_{i(D)}-k\right)} \\
&\quad + RT(C_k))) + ((2^{n_{\mathrm{i(D)}}} - 1) \times \mathrm{N}))
\end{aligned} \tag{7}$$

Based on Equation (7), in the worst case, the time complexity of the Incremental-CUF-growth is $O(2^{n_{\mathrm{i(D)}}} \times \mathrm{N})$ by considering only the highest order term. As a result, since $O(2^{n_{\mathrm{i(D)}}} \times \mathrm{N}) > O(2^{n_{\mathrm{i(D)}}})$, the proposed ILUNA algorithm can finish its mining process much faster than Incremental-CUF-growth, theoretically.

5. Performance evaluation

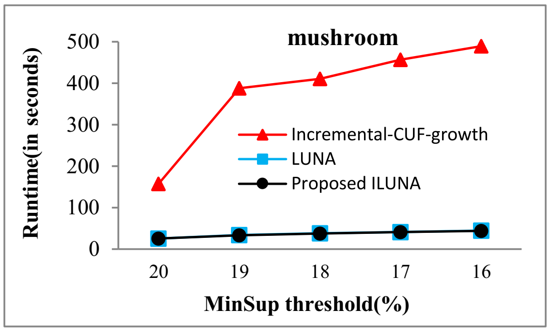
In this section, the results of the comprehensive experimental analysis of the performance of the proposed ILUNA algorithm compared to the LUNA and Incremental-CUF-growth algorithms are shown for several real and synthetic datasets. Table 5 illustrates some statistical information of datasets applied in the experimental analysis. Indeed, these datasets are famous datasets widely used in the frequent pattern mining field. The synthetic sparse datasets T10I4D100K, T40I10D100K, u12I5D300K, and T10I5D1M have been obtained through the IBM Quest Synthetic Data Generator [49] and two dense real-life datasets mushroom and Connect-4 and three sparse real-life datasets Pumsb*, Retail, and Accidents have been obtained from the FIMI Repository [50]. The fourth and fifth columns illustrate the average and maximal length of transactions, respectively, and the degree of density or sparsity of datasets is shown in the sixth column. Each dataset is labeled based on some of its characteristics. For example, the dataset T40I10D100K contains 100 K transactions and its average transaction length is 40. In these datasets, each item in every transaction has been assigned to an (randomly generated) existential probability value in (0, 1] range [26,42,45]. All the programs have been implemented by C# code and run with Windows 10 as the operating system on a personal 64-bit laptop with Intel(R) Core(TM) i7-6700 HQ CPU 2.60 GHz, 16 GB RAM. The results reported in this section have been obtained from the average of multiple runs for each case and in all experiments, the minSup threshold has been represented as the percentage of uncertain database size. In these experiments, the proposed ILUNA, LUNA, and Incremental-CUF-growth algorithms have been compared in terms of the execution time and the scalability with respect to the number of transactions. To avoid unfairness and take into account all the conditions, the runtime evaluations are divided into four sets; updating in the worst case, updating in the best case, regular updating, and irregular updating.

5.1. Updating in the worst case

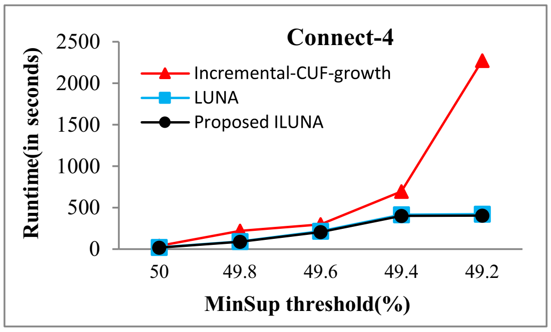
The worst case for incremental updating is defined as the state in which the original database UDB_0 is smaller than the incremental database UDB_i because in this case the incremental algorithms slowly extract the UFPs due to computational overheads. In this study for the worst case, 20% of the database is considered as UDB_0 and 80% of the remaining database is added as incremental database UDB_1. As seen in Fig. 10, the effects of minSup changes on the runtime in the worst case have been investigated for the LUNA, Incremental-CUF-growth, and proposed ILUNA algorithms on datasets mushroom, connect-4, Retail, Pumsb*, Accidents, T10I4D100K, T40I10D100K, and u12I5D300K. By comparing the evaluation results of the ILUNA and LUNA algorithms, it can be concluded that in the worst case, the proposed algorithm has more computational overheads when mining the UFPs from dense datasets than sparse ones. Therefore, it can extract the UFPs only 5% faster than the LUNA algorithm in minSup 49.2% on the Connect-4 dataset. However, despite the overload of computations in the worst

Table 5
Dataset characteristics.

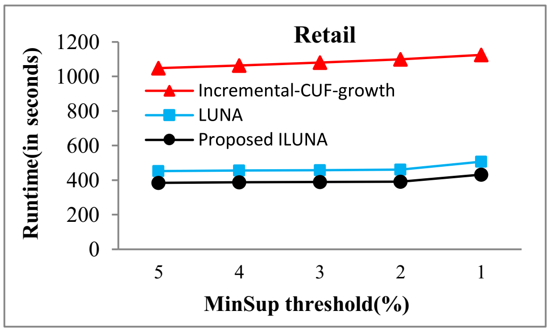
Dataset	#Trans	#Items	Trans Len		Ave/#Items(×100)	Dense/Sparse
			Ave	Max		
mushroom	8124	119	23	23	19.33	Dense
Connect-4	67,557	129	43	43	33.33	Dense
Pumsb*	49,046	2088	50.48	63	2.42	Sparse
Retail	88,162	16,470	10.31	76	0.06	Sparse
Accidents	340,183	468	33.8	51	7.22	Sparse
T10I4D100K	100,000	870	10.10	29	1.16	Sparse
T40I10D100K	100,000	942	39.61	77	4.20	Sparse
u12I5D300K	300,000	1000	12.25	31	1.225	Sparse
T10I5D1M	1,000,000	1000	10	32	1	Sparse



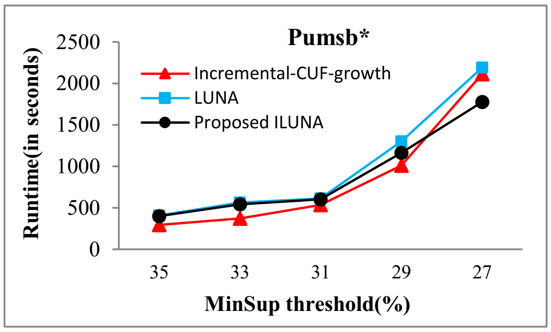
(a) mushroom dataset



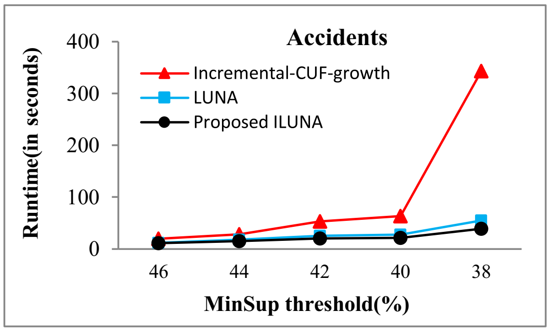
(b) Connect-4 dataset



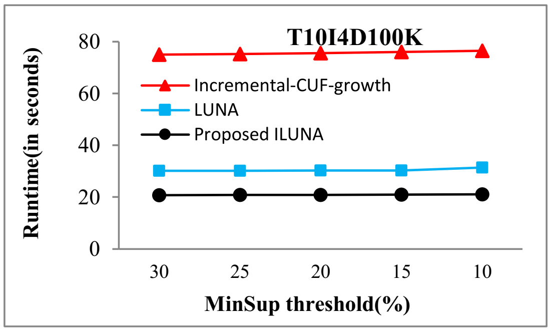
(c) Retail dataset



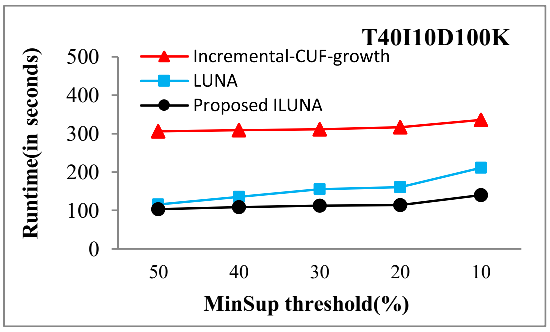
(d) Pumsb* dataset



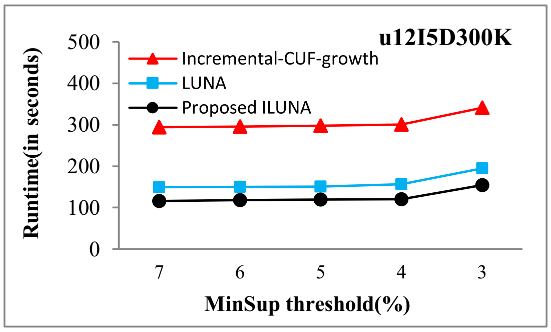
(e) Accidents dataset



(f) T10I4D100K dataset



(g) T40I10D100K dataset



(h) u12I5D300K dataset

Fig. 10. Experimental results: The RunTime of updating in the worst case.

case, the proposed algorithm performs well in extracting the UFPs from sparse datasets. For example, the ILUNA algorithm extracts the UFPs approximately 33% faster than the LUNA algorithm in minSup 10% on the T10I4D100K dataset. Comparing the LUNA and Incremental-CUF-growth reveals that although the Incremental-CUF-growth algorithm has been designed for dynamic environments, it requires more time than the static LUNA algorithm in most datasets because it produces a lot of false positives in low minSups that take time to filter. Comparing the evaluation results of the two algorithms ILUNA and Incremental-CUF-growth shows that even in the worst case, the proposed method achieves more impressive performance than the Incremental-CUF-growth in all datasets. The reasons for the great performance of the proposed method can be as follows: (i) when an incremental database UDB_i is added, the ILUNA updates both IUP-Lists and ICUP-Lists. Indeed, the mining process information of the previous stages have been stored in the ICUP-Lists and the proposed algorithm only updates them. Therefore, the algorithm does not need to repeat the mining process from scratch. However, the Incremental-CUF-growth only updates its global tree and has to do the process of constructing conditional trees and mining potential patterns from scratch; (ii) The ILUNA extracts exact results of uncertain frequent pattern mining, while the Incremental-CUF-growth generates a lot of false positives that take time to build conditional trees for them; (iii) The ILUNA algorithm scans only the new database (UDB_1), while the Incremental-CUF-growth has to scan UDB_1 and rescan the entire database (UDB_0 + UDB_1) to filter out false positives from the result set. At low minSups where many false positives are produced, this additional scan is very time-consuming. For example, the ILUNA algorithm extracts the UFPs approximately 91% faster than the Incremental-CUF-growth algorithm in minSup 16% on the mushroom dataset.

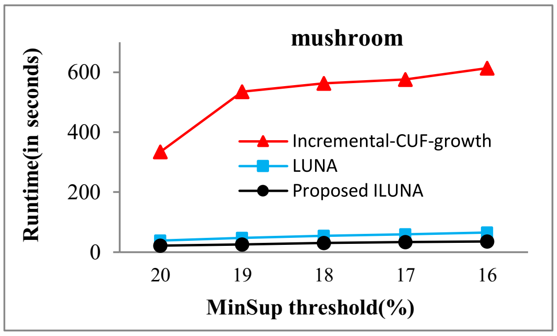
It is important to note that the overall performance of upper bound-based algorithms strongly depends on the number of generated false positives [46]. Indeed, the main computational cost of the proposed algorithm is spent on comparing lists to build or update them, while the step that has the highest computational cost in the Incremental-CUF-growth algorithm corresponds to filtering the false positives. In some datasets such as pumsb* in high minSups where fewer patterns are extracted, the Incremental-CUF-growth algorithm generates few false positives. Therefore, it does not require much time to filter them and can use less running time than the proposed algorithm. However, as the minSup is decreased, more patterns are extracted and the number of generated false positives by the Incremental-CUF-growth algorithm is increased that takes a long time to be filtered. Therefore, the proposed algorithm excels in low minSups.

5.2. Updating in the best case

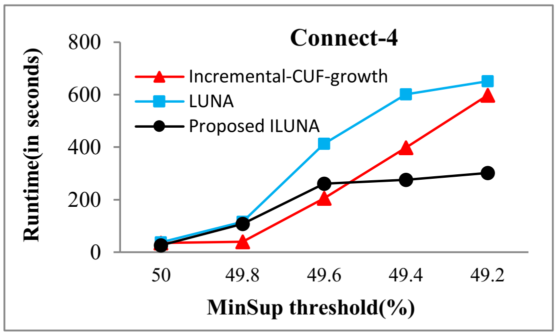
The best case for incremental updating is defined as the state in which the original database UDB_0 is larger than the incremental database UDB_i because in this case, the incremental algorithms have the highest efficiency. In this study for the best case, 80% of the database is considered as UDB_0 and the remaining 20% of the database is added as incremental database UDB_1. The efficiency of the algorithms in the best case has been evaluated with minSup changes on datasets mushroom, connect-4, Retail, Pumsb*, Accidents, T10I4D100K, T40I10D100K, and u12I5D300K. As can be seen from the results of the evaluations in Fig. 11, for all datasets, the proposed algorithm extracts the UFPs much faster than the LUNA and Incremental-CUF-growth algorithms. Indeed, after adding the database UDB_1 to UDB_0, the proposed algorithm only scans the UDB_1 once and updates the IUP-Lists and ICUP-Lists for UDB_1. However, the LUNA algorithm must scan the entire database (UDB_0 + UDB_1) twice and construct the UP-Lists and CUP-Lists from scratch. Moreover, the Incremental-CUF-growth algorithm has to repeat the mining process from scratch and rescan the entire database (UDB_0 + UDB_1) to filter out false positives. Therefore, the LUNA and Incremental-CUF-growth algorithms need more time to extract the UFPs. Especially in low minSups where more UFPs are extracted, the gap between the proposed ILUNA and other algorithms is increased. For example, the proposed algorithm extracts the UFPs approximately 58% and 94% faster than the LUNA and Incremental-CUF-growth algorithms in minSup 38% on the Accidents dataset, respectively.

5.3. Regular updating

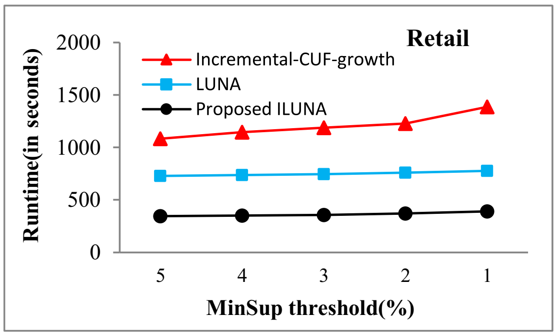
The regular updating refers to a situation in which the number of transactions of the original and incremental databases is equal. Hence, in the third experimental set, to conduct the performance evaluation of the LUNA, Incremental-CUF-growth, and ILUNA algorithms on incremental data processing environments, each database UDB has been divided into five equal portions (each portion includes 20% of the entire database). The first portion is considered as the original database (UDB_0) and four other portions are regarded as the incremental databases (UDB_i). At each stage, a portion is incrementally added to the previous database and the effects of incremental updating on runtime are evaluated. Experimental results for datasets mushroom with minSup threshold = 20%, connect-4 with minSup threshold = 49.2%, Retail with minSup threshold = 5%, Pumsb* with minSup threshold = 31%, Accidents with minSup threshold = 36%, T10I4D100K with minSup threshold = 10%, T40I10D100K with minSup threshold = 10%, and u12I5D300K with minSup threshold = 3% is shown in Fig. 12. As the results of the experiment show, by increasing the number of updates, all algorithms need more runtime to extract patterns. For UDB_0, the LUNA algorithm requires two UDB_0 scans to extract UFPs but the ILUNA algorithm extracts UFPs with a single UDB_0 scan. Therefore, the LUNA algorithm needs more time to scan the database. On the contrary, the proposed algorithm constructs the IUP-Lists for frequent and infrequent patterns while the LUNA algorithm creates the UP-Lists for only frequent patterns. As a result, for the UDB_0, for cases where the time difference required to construct UP-Lists and IUP-Lists is greater than the time difference required to scan the UDB_0 by LUNA and ILUNA, the proposed ILUNA algorithm may require more time to extract the UFPs. However, when the incremental database UDB_i is added, the proposed ILUNA extracts patterns



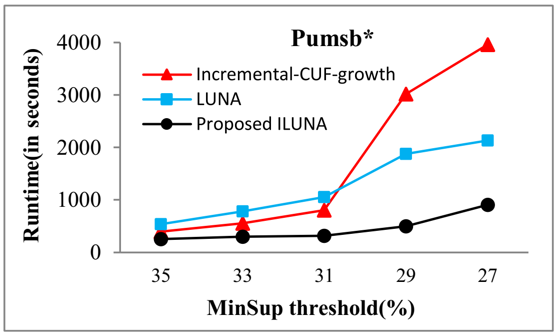
(a) mushroom dataset



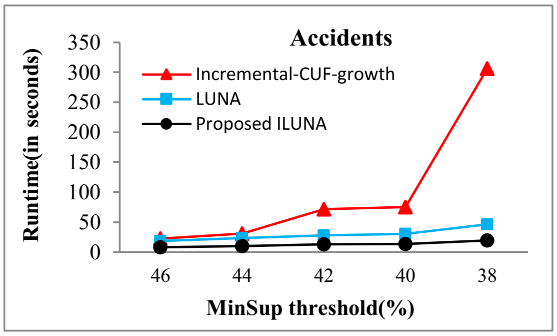
(b) Connect-4 dataset



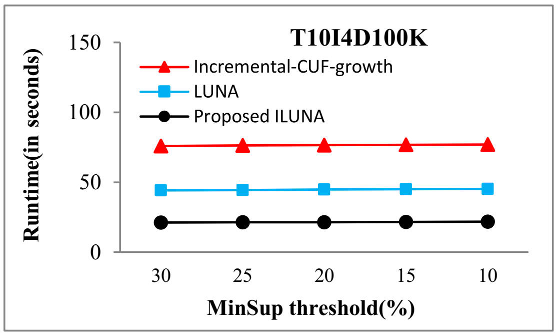
(c) Retail dataset



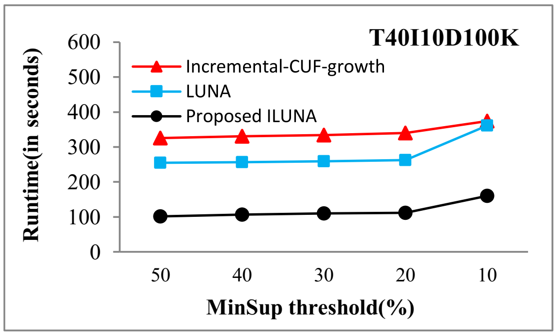
(d) Pumsb* dataset



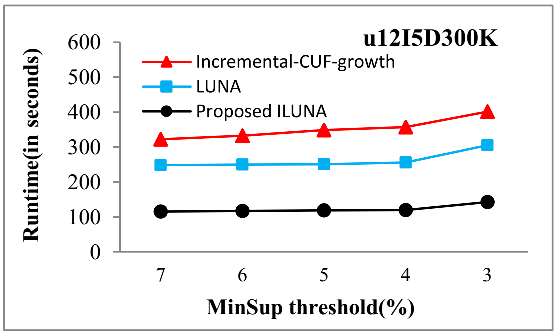
(e) Accidents dataset



(f) T10I4D100K dataset

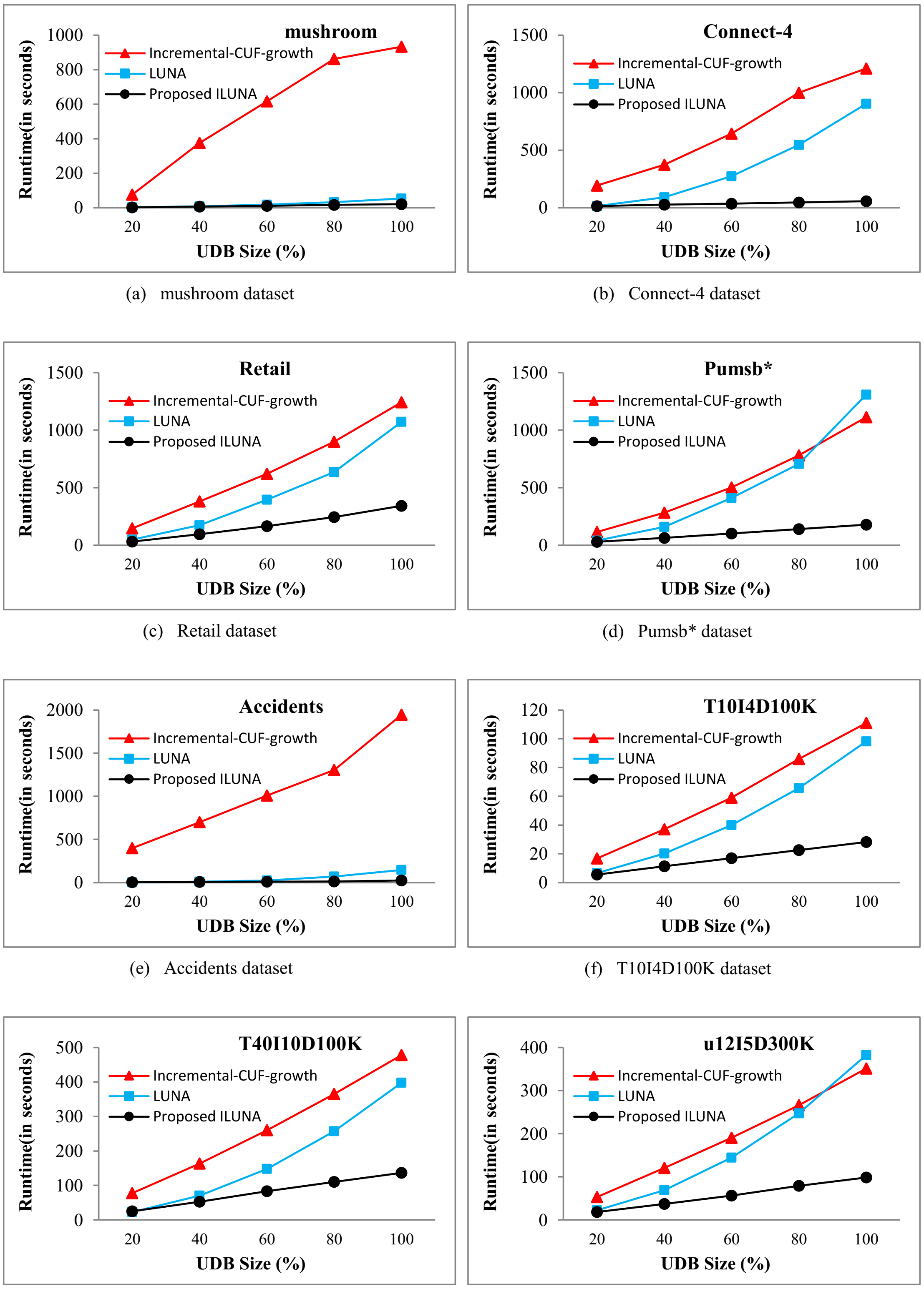


(g) T40I10D100K dataset



(h) u12I5D300K dataset

Fig. 11. Experimental results: RunTime of updating in the best case.



(a) mushroom dataset

(b) Connect-4 dataset

(c) Retail dataset

(d) Pumsb* dataset

(e) Accidents dataset

(f) T10I4D100K dataset

(g) T40I10D100K dataset

(h) u12I5D300K dataset

Fig. 12. Experimental results: The RunTime of regular updating.

in much less time because the LUNA algorithm should scan the whole database (UDB_0 + UDB_i) twice while the proposed algorithm only scans the UDB_i once. On the other hand, the LUNA algorithm has to repeat the whole process of extracting the patterns, including generating all the UP-Lists and CUP-Lists, from the scratch. However, the proposed ILUNA algorithm has stored the IUP-Lists and ICUP-Lists of the previous stages and only updates them. As the evaluation results show, the Incremental-CUF-growth algorithm requires much more time than the proposed algorithm because whenever a new database is added, it needs an additional scan to filter false positives, which is very time consuming, especially for low minSups in large databases with more transactions and items. Hence, as the number of update portions is increased, the gap between the proposed ILUNA and other algorithms is increased. For instance, in the last stage of the update process in the Accidents dataset (Fig. 12(e)), the required runtimes for algorithms LUNA and Incremental-CUF-growth are 146.36 and 1945.985 s, respectively but the ILUNA only needs 23.846 s. In other words, the proposed ILUNA algorithm extracts the UFPs approximately 84% and 99% faster than the LUNA and Incremental-CUF-growth algorithms, respectively.

5.4. Irregular updating

Unlike regular updating, in irregular updating, the number of transactions of the original and incremental databases is unequal. In fact, in most cases in the real world, the number of data added to a database does not have a specific regularity. Hence, in the fourth experimental set, the results of irregular updating for datasets mushroom with minSup threshold = 20%, connect-4 with minSup threshold = 49.2%, Retail with minSup threshold = 5%, Pumsb* with minSup threshold = 31%, Accidents with minSup threshold = 36%, T10I4D100K with minSup threshold = 10%, T40I10D100K with minSup threshold = 10%, and u12I5D300K with minSup threshold = 3% is shown in Figs. 13 and 14. These experiments focus on two categories of irregular updating. In the first category, the number of transactions of the original database is unequal to the number of transactions of incremental databases; however, the incremental databases have an equal number of transactions. To perform this assessment, each UDB has been divided into four update portions (70%-10%-10%-10%). Initially, one portion includes 70% of the database as UDB_0 is entered. Then, another 10% of the database is added as UDB_1, and patterns are extracted from them. Next, in the same way, another 10% of the database as UDB_2, and finally the remaining 10% of the database as UDB_3 are added to the previous data, and evaluations are performed. As the results of the evaluations show in Fig. 13, with the addition of each new database, the proposed algorithm can provide more runtime performance. For instance, after the third update in the Pumsb* dataset (Fig. 13(d)), the proposed ILUNA algorithm extracts the UFPs approximately 80% and 83% faster than the LUNA and Incremental-CUF-growth algorithms, respectively.

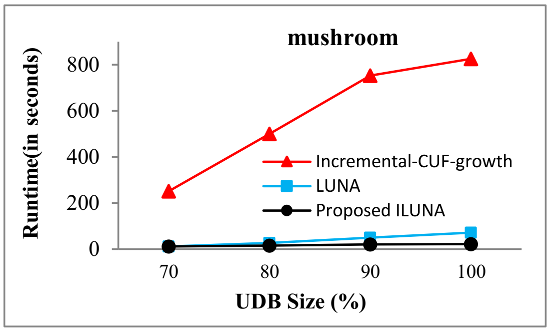
In the second category, all the original and incremental databases have an unequal number of transactions. For this reason, each UDB has been divided into four update portions (20–10–40–30%). Therefore, one portion includes 20% of the database as UDB_0 is entered and three other portions include 10%, 40%, and 30% of the database as UDB_1, UDB_2, and UDB_3, respectively, are added to the previous data. The results of the evaluations can be seen in Fig. 14. In this category, the proposed algorithm has also high efficiency for incremental mining of UFPs. For instance, after the third update in the connect-4 dataset (Fig. 14(b)), the proposed ILUNA algorithm extracts the UFPs approximately 89% and 95% faster than the LUNA and Incremental-CUF-growth algorithms, respectively.

5.5. Scalability evaluation

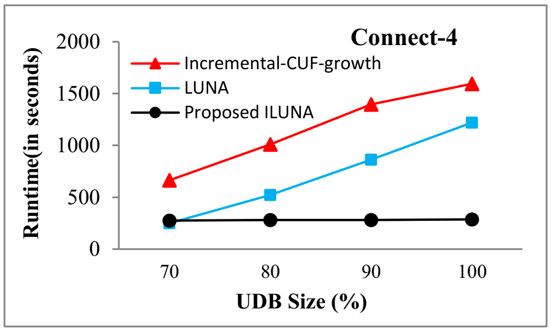
In this section, the scalability of the LUNA, Incremental-CUF-growth, and ILUNA algorithms is evaluated by varying the number of transactions in the database at runtime. To evaluate the scalability, T10I5D1M is used which is a huge sparse dataset with one million transactions. It is divided into five portions of 0.2 million transactions and the minSup is fixed at 20%. The number on the y-axis shows the runtime required to extract UFPs and the x-axis shows the number of transactions. As the evaluation result in Fig. 15 shows, the runtime required to extract patterns is increased with increasing the size of the dataset. Therefore, the proposed algorithm has significant scalability compared to the LUNA and Incremental-CUF-growth algorithms in the large-scale data.

6. Conclusion and future works

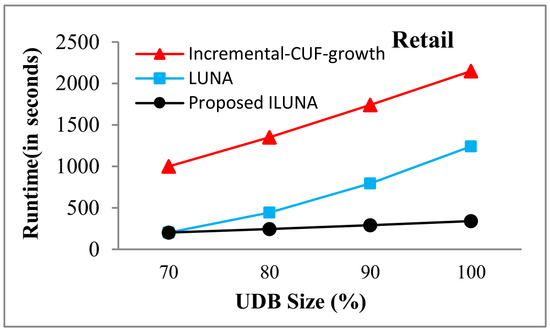
In this paper, an efficient method for incremental mining of frequent patterns from uncertain data has been proposed. This method extracts the frequent patterns without any false positives with only a single database scan and manages to store the information of lists well by proposing two new data structures IUP-List and ICUP-List. Accordingly, when the original database UDB_0 is updated and an incremental database UDB_i is added, it only updates the IUP-Lists and ICUP-Lists by a single scan of the UDB_i. However, static LUNA algorithm has to repeat the whole mining process from scratch by scanning the entire database (UDB_0 + UDB_i) twice and dynamic Incremental-CUF-growth algorithm has to eliminate generated false positives by an additional scan of the entire database (UDB_0 + UDB_i). Extensive performance evaluations reveal that the ILUNA algorithm extracts the UFPs much faster than the LUNA and Incremental-CUF-growth algorithms in various types of updating modes. Additionally, it considerably performs better than the LUNA and Incremental-CUF-growth algorithms in terms of scalability. In future works, another functionality such as interactive capability can be added to the proposed algorithm. In interactive environments, the database remains unchanged and the user extracts new frequent patterns by changing the minSup thresh-



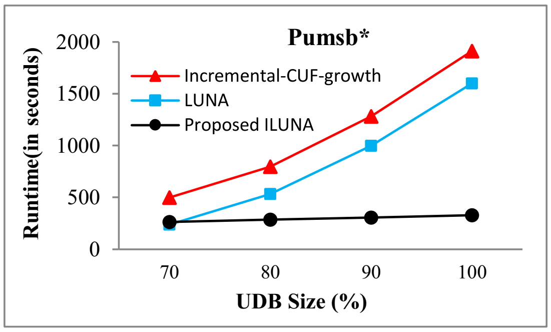
(a) mushroom dataset



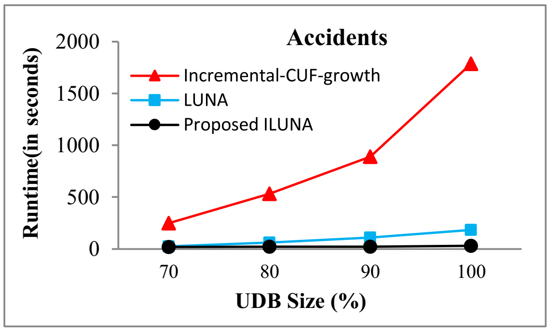
(b) Connect-4 dataset



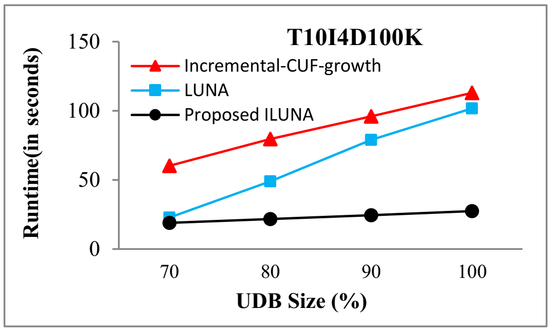
(c) Retail dataset



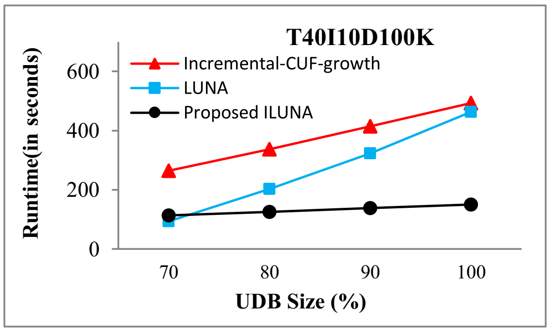
(d) Pumsb* dataset



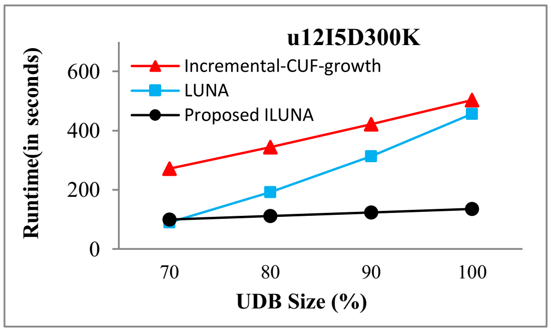
(e) Accidents dataset



(f) T10I4D100K dataset

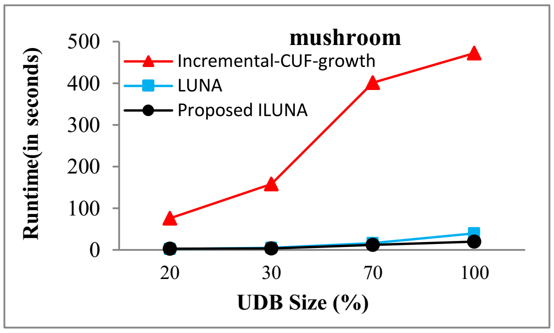


(g) T40I10D100K dataset

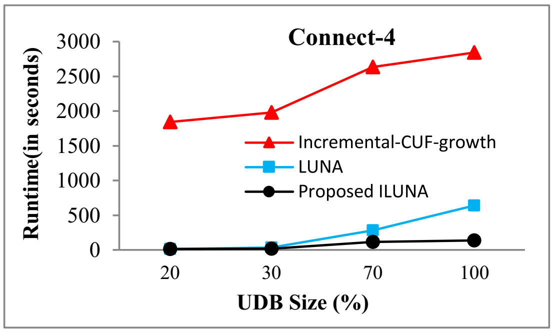


(h) u12I5D300K dataset

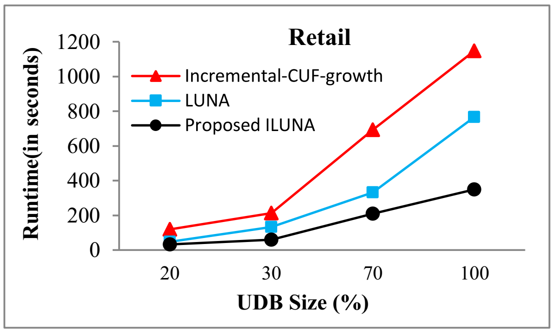
Fig. 13. Experimental results: The RunTime of category1 of irregular updating.



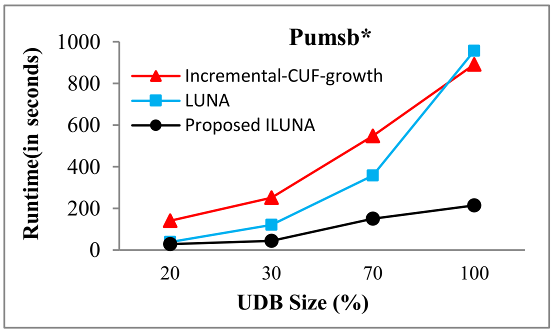
(a) mushroom dataset



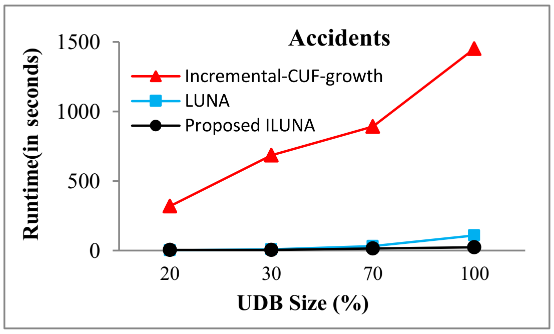
(b) Connect-4 dataset



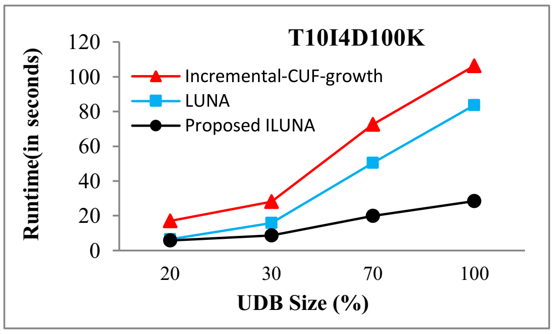
(c) Retail dataset



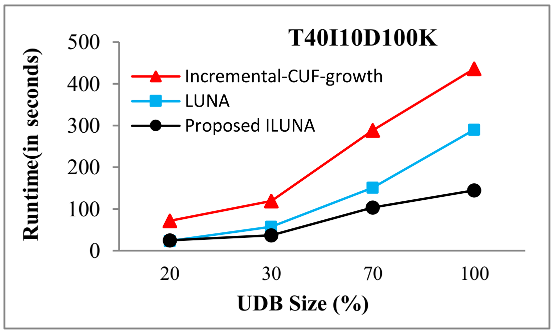
(d) Pumsb* dataset



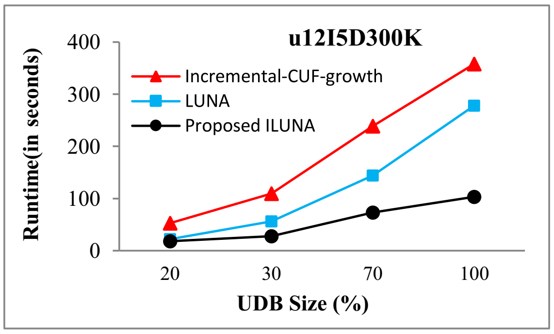
(e) Accidents dataset



(f) T10I4D100K dataset



(g) T40I10D100K dataset



(h) u12I5D300K dataset

Fig. 14. Experimental results: The RunTime of category2 of irregular updating.

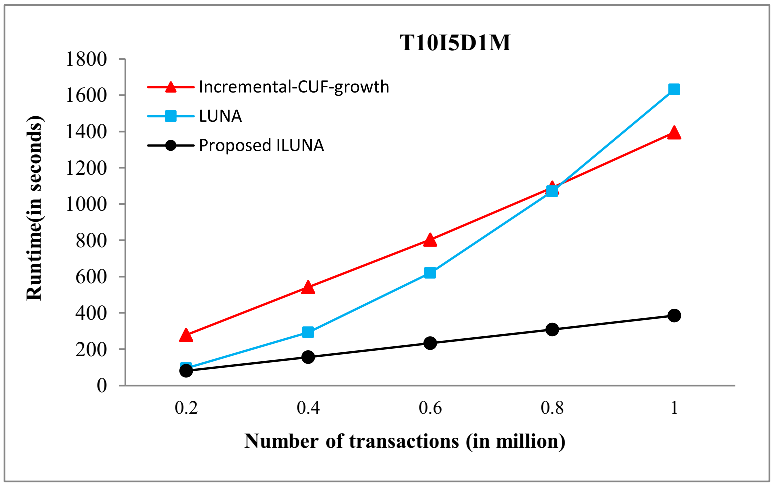


Fig. 15. Experimental results: Scalability with No. of transactions.

old. Indeed, future research might be an attempt to provide an algorithm that can easily extract the UFPs without repeating the mining process from scratch when the minSup threshold changes by the user.

CRediT authorship contribution statement

Razieh Davashi: Conceptualization, Data curation, Formal analysis, Investigation, Methodology, Project administration, Resources, Software, Supervision, Validation, Visualization, Writing - original draft, Writing - review & editing.

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

References

[1] J.-P. Qiang, P. Chen, W. Ding, F. Xie, X. Wu, Multi-document summarization using closed patterns, Knowl.-Based Syst. 99 (2016) 28–38.
[2] M.R. Karim, M. Cochez, O.D. Beyan, C.F. Ahmed, S. Decker, Mining maximal frequent patterns in transactional databases and dynamic data streams: a spark-based approach, Inf. Sci. 432 (2018) 278–300.
[3] H.M. Huynh, L.T. Nguyen, B. Vo, A. Nguyen, V.S. Tseng, Efficient methods for mining weighted clickstream patterns, Expert Syst. Appl. 142 (2020) 112993.
[4] H. Nam, U. Yun, E. Yoon, J.C.W. Lin, Efficient approach for incremental weighted erasable pattern mining with list structure, Expert Syst. Appl. 143 (2020) 113087.
[5] X. Dong, Y. Gong, L. Cao, F-NSP+: A fast negative sequential patterns mining method with self-adaptive data storage, Pattern Recogn. 84 (2018) 13–27.
[6] J. Zhang, X. Zhao, S. Zhang, S. Yin, X. Qin, Interrelation analysis of celestial spectra data using constrained frequent pattern trees, Knowl.-Based Syst. 41 (2013) 77–88.
[7] H. Nam, U. Yun, E. Yoon, J. Chun- Wei Lin, Efficient approach of recent high utility stream pattern mining with indexed list structure and pruning strategy considering arrival times of transactions, Inf. Sci. 529 (2020) 1–27.
[8] A. Dhiman, S.K. Jain, Optimizing frequent subgraph mining for single large graph, Procedia Comput. Sci. 89 (2016) 378–385.
[9] X. Wang, Y. Xu, H. Zhan, Extending association rules with graph patterns, Expert Syst. Appl. 141 (2020) 112897.
[10] B. Vo, T. Le, W. Pedrycz, G. Nguyen, S.W. Baik, Mining erasable itemsets with subset and superset itemset constraints, Expert Syst. Appl. 69 (2017) 50–61.
[11] G. Lee, U. Yun, Single-pass based efficient erasable pattern mining using list data structure on dynamic incremental databases, Future Generation Computer Systems 80 (2018) 12–28.
[12] H. Chen, Mining top-k frequent patterns over data streams sliding window, J Intell Inf Syst 42 (1) (2014) 111–131.
[13] J. Ashraf, A. Habib, A. Salam, Top-k miner: top-k identical frequent itemsets discovery without user support threshold, Knowl. Inf. Syst. 48 (3) (2016) 741–762.
[14] U. Yun, G. Lee, Sliding window based weighted erasable stream pattern mining for stream data applications, Future Generation Computer Systems 59 (2016) 1–20.
[15] R. Agarwal, R. Srikant, Fast algorithms for mining association rules, in, in: Proceedings of the 20th International Conference very large data bases, 1994, pp. 487–499.
[16] J. Han, J. Pei, Y. Yin, Mining frequent patterns without candidate generation, SIGMOD Rec. 29 (2) (2000) 1–12.
[17] C.K.S. Leung, Q.I. Khan, Z. Li, T. Hoque, CanTree: a canonical-order tree for incremental frequent-pattern mining, Knowl. Inf. Syst. 11 (3) (2007) 287–311.
[18] S.K. Tanbeer, C.F. Ahmed, B.-S. Jeong, Y.-K. Lee, Efficient single-pass frequent pattern mining using a prefix-tree, Inf. Sci. 179 (5) (2009) 559–583.
[19] C.-W. Lin, T.-P. Hong, W.-H. Lu, The Pre-FUFP algorithm for incremental mining, Expert Syst. Appl. 36 (5) (2009) 9498–9505.
[20] Y.S. Koh, G. Dobbie, Efficient single pass ordered incremental pattern mining, Trans. Large-Scale Data- Knowl.-Centered Syst. 8 (1) (2013) 137–156.

[21] R. Davashi, M.H. Nadimi-Shahraki, EFP-tree: an efficient FP-tree for incremental mining of frequent patterns, Int. J. Data Mining, Modelling Manage. 11 (2) (2019) 144–166.
[22] C.-K. Chui, B. Kao, E. Hung, Mining frequent itemsets from uncertain data, in: Z.-H. Zhou, H. Li, Q. Yang (Eds.), Lecture Notes in Computer ScienceAdvances in Knowledge Discovery and Data Mining, Springer Berlin Heidelberg, Berlin, Heidelberg, 2007, pp. 47–58.
[23] C.K.S. Leung, C.L. Carmichael, B. Hao, Efficient mining of frequent patterns from uncertain data, in: Proceedings of the International Conference on Data Mining Workshops, 2007, pp. 489–494.
[24] C.K.S. Leung, M.A.F. Mateo, D.A. Brajczuk, A tree-based approach for frequent pattern mining from uncertain data, in, in: Proceedings of the Pacific-Asia Conference on Knowledge Discovery and Data Mining, 2008, pp. 653–661.
[25] T.L. Lin, B.W. Wen, H.Y. Chang, W.K. Chang, S.C. Hsu, A Rapid Incremental Frequent Pattern Mining Algorithm for Uncertain Data, in: Proceedings of the 2017 5th Intl Conf on Applied Computing and Information Technology/4th Intl Conf on Computational Science/Intelligence and Applied Informatics/ 2nd Intl Conf on Big Data, Cloud Computing, Data Science (ACIT-CSII-BCD), IEEE, 2017, pp. 284–288.
[26] C.K.S. Leung, S.K. Tanbeer, Fast tree-based mining of frequent itemsets from uncertain data, in, in: Proceedings of the 17th International Conference on Database Systems for Advanced Applications, 2012, pp. 272–287.
[27] G. Lee, U. Yun, A new efficient approach for mining uncertain frequent patterns using minimum data structure without false positives, Future Generation Computer Systems 68 (2017) 89–110.
[28] G. Pyun, U. Yun, K.H. Ryu, Efficient frequent pattern mining based on Linear Prefix tree, Knowl.-Based Syst. 55 (2014) 125–139.
[29] Y. Djenouri, D. Djenouri, A. Belhadi, A. Cano, Exploiting GPU and cluster parallelism in single scan frequent itemset mining, Inf. Sci. 496 (2019) 363–377.
[30] Y. Djenouri, J.C.W. Lin, K. Nørvåg, H. Ramampiaro, Highly efficient pattern mining based on transaction decomposition, in: Proceedings of the IEEE International Conference on Data Engineering, 2019, pp. 1646–1649.
[31] U. Ahmed, J.-W. Lin, G. Srivastava, R. Yasin, Y. Djenouri, An Evolutionary Model to Mine High Expected Utility Patterns From Uncertain Databases, IEEE Trans. Emerg. Top. Comput. Intell. 5 (1) (2021) 19–28.
[32] C.-K. Chui, B. Kao, A decremental approach for mining frequent itemsets from uncertain data, in: T. Washio, E. Suzuki, K.M. Ting, A. Inokuchi (Eds.), Lecture Notes in Computer ScienceAdvances in Knowledge Discovery and Data Mining, Springer Berlin Heidelberg, Berlin, Heidelberg, 2008, pp. 64–75.
[33] L. Wang, D.-L. Cheung, R. Cheng, S.D. Lee, X.S. Yang, Efficient Mining of Frequent Item Sets on Large Uncertain Databases, IEEE Trans. Knowl. Data Eng. 24 (12) (2012) 2170–2183.
[34] X. Sun, L. Lim, S. Wang, An Approximation Algorithm Of Mining Frequent Itemsets From Uncertain Dataset, IJACT 4 (3) (2012) 42–49.
[35] T. Calders, C. Garboni, B. Goethals, Efficient pattern mining of uncertain data with sampling, in: Proceedings of the 14th Pacific-Asia Conference on Knowledge Discovery and Data Mining, 2010, pp. 480–487.
[36] L.A. Abd-Elmegid, M.E. El-Sharkawi, L.M. El-Fangary, Y.K. Helmy, Vertical mining of frequent patterns from uncertain data, Computer Inform. Sci. 3 (2) (2010) 171–179.
[37] C.K.S. Leung, L. Sun, Equivalence class transformation based mining of frequent itemsets from uncertain data, in: Proceedings of the 2011 ACM Symposium on Applied Computing, 2011, pp. 983–984.
[38] C.K.S. Leung, S.K. Tanbeer, B.P. Budhia, L.C. Zacharias, Mining probabilistic datasets vertically, in: Proceedings of the 16th International Database Engineering & Applications Sysmposium, 2012, pp. 199–204.
[39] M.J. Zaki, S. Parthsarathy, M. Ogihara, W. Li, New algorithms for fast discovery of association rules, in: Proceedings of the International Conference on Knowledge Discovery and Data Mining, 1997, pp. 283–286.
[40] C.C. Aggarwal, Y. Li, J. Wang, J. Wang, Frequent pattern mining with uncertain data, in: Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining, 2009, pp. 29–37.
[41] R.S. Bhadoria, R. Kumar, M. Dixit, Analysis on probabilistic and binary datasets through frequent itemset mining, in: Proceedings of 2011 World Congress on Information and Communication Technologies, 2011, pp. 263–267.
[42] C. K. S. Leung, S. K. Tanbeer, PUF-tree: a compact tree structure for frequent pattern mining of uncertain data, in: Proceedings of the 17th Pacific-Asia Conference on Knowledge Discovery and Data Mining, PAKDD, Part I, 2013, pp. 13-25.
[43] R. K. MacKinnon, T. D. Strauss, C. K. S. Leung, DISC: efficient uncertain frequent pattern mining with tightened upper bounds, in: Proceedings of the IEEE ICDM Workshops 2014. IEEE Computer Society, 2014, pp. 1038-1045.
[44] C. K. S. Leung, R. K. MacKinnon, BLIMP: a compact tree structure for uncertain frequent pattern mining, in: Proceedings of the International Conference on Data Warehousing and Knowledge Discovery, 2014, pp. 115-123.
[45] C.K. Leung, R.K. MacKinnon, S.K. Tanbeer, Tightening Upper Bounds to the Expected Support for Uncertain Frequent Pattern Mining, Procedia Comput. Sci. 35 (2014) 328–337.
[46] C.K.S. Leung, R.K. MacKinnon, Balancing tree size and accuracy in fast mining of uncertain frequent patterns, in: Proceedings of the 17th International Conference on Big Data Analytics and Knowledge Discovery, 2015, pp. 57–69.
[47] C.-W. Lin, T.-P. Hong, A new mining approach for uncertain databases using CUFP trees, Expert Syst. Appl. 39 (4) (2012) 4084–4093.
[48] L. Wang, L. Feng, M. Wu, AT-mine: an efficient algorithm of frequent itemset mining on uncertain dataset, J. Computers 8 (6) (2013) 1417–1426.
[49] R. Agrawal, R. Srikant, Quest synthetic data generator, http://www.Almaden.ibm.com/cs/quest/syndata.html.
[50] Frequent itemset mining dataset repository, http://fimi.ua.ac.be/data/.