Information Sciences 615 (2022) 1–23

Contents lists available at ScienceDirect

Information Sciences

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

# An efficient approach for mining weighted uncertain interesting patterns

Ham Nguyen [a], Dang Vo [a], Huong Bui [b], Tuong Le [c,d,*], Bay Vo [a]

[a] Faculty of Information Technology, HUTECH University, Ho Chi Minh City, Vietnam
[b] Faculty of Computing Fundamentals, FPT University, Ho Chi Minh City, Vietnam
[c] Informetrics Research Group, Institute for Computational Science, Ton Duc Thang University, Ho Chi Minh City, Vietnam
[d] Faculty of Information Technology, Ton Duc Thang University, Ho Chi Minh City, Vietnam

ARTICLE INFO

*Article history:*
Received 3 November 2021
Received in revised form 26 July 2022
Accepted 3 October 2022
Available online 10 October 2022

*Keywords:*
Data mining
Pattern mining
Weighted uncertain interesting pattern
TPN-list structure

ABSTRACT

The mining of weighted uncertain interesting patterns (WUIPs) is an interesting problem with many practical applications. In this study, we propose an effective method for mining WUIPs from uncertain databases. We first introduce the TPN-list structure, an extended version of the *N*-list structure, which is used to represent and discover the WUIPs. A TPN-list intersection algorithm is then developed, which has linear complexity and the ability to self-reduce its size. Several theorems are also proposed for the fast calculation and determination of a WUIP based on its TPN-list. Finally, we propose the HWUIPM algorithm, based on the above proposals, for mining WUIPs from uncertain databases. Our experimental results demonstrate that the proposed algorithm outperforms other state-of-the-art algorithms for mining WUIPs in terms of running time, resource usage and scalability.



## 1. Introduction

Data mining and machine learning are becoming increasingly important in the era of Industry 4.0 and have attracted many research scientists. They can help systems to make intelligent decisions based on data from many different sources. In the data mining domain, many types of patterns have been proposed, such as frequent [1,18], erasable [36], and utility patterns [37], which can be used in different specific cases. Of these, frequent patterns are the most famous, with many algorithms such as PrePost [12], PrePost$^+$ [11], NSFI [35], PSO-Apriori [14], and numerous applications [4]. The frequent weighted pattern (FWP), a variation of a frequent pattern that allows for different levels of significance or weights of items in a database, also been of interest recently. Nguyen et al. [33] proposed an efficient algorithm for mining frequent weighted itemsets using interval word segments, while in 2017, Lee et al. [24] introduced an approach for mining FWPs without the need to store transaction IDs and generate candidates. Bui et al. [6] presented an efficient method for mining frequent weighted closed itemsets, and Lin et al. [29] developed an approach for mining FWPs from uncertain databases.

The problem of mining uncertain frequent patterns [2] arises when noisy data sources are used, such as acoustic, electromagnetic, or optical radiation, wireless sensors, wifi systems, mechanical systems, GPS in environmental surveillance, secu-

Peer review under responsibility of For special issue article please include a footnote that this paper belongs to the special issue "special issue name" edited by "editor name".

* Corresponding author at: Ton Duc Thang University, Ho Chi Minh City, Vietnam.
*E-mail addresses:* nd.ham@hutech.edu.vn (H. Nguyen), huongbd2@fpt.edu.vn (H. Bui), lecungtuong@tdtu.edu.vn (T. Le), vd.bay@hutech.edu.vn (B. Vo).

https://doi.org/10.1016/j.ins.2022.10.009
0020-0255/

rity, chemical, and manufacturing systems. These data sources contain errors for several reasons, such as inherent inaccuracies of measurement, the sampling frequency of the sensors, deviations caused by rapid changes to the measured property over time, wireless transmission errors, or network latencies. There are many algorithms that have been developed for mining useful patterns in uncertain databases, such as uncertain frequent patterns [2,9,25,26,27,28], top-*k* uncertain frequent patterns [22], uncertain maximal frequent patterns [16], uncertain weighted frequent itemsets [30], uncertain recent weighted-based patterns [17], uncertain weighted frequent sequences [34], uncertain high-utility itemsets [7,31,32], high utility-occupancy patterns [8], high expected utility patterns [38], and weighted uncertain interesting patterns [3]. Recently, Ahmed et al. [3] proposed a new strategy for mining weighted uncertain interesting patterns (WUIPs) from uncertain databases in which the WUIPs found by the algorithm could represent correlations between items with different importance values. An WUIP is a pattern that satisfies three factors: frequency, correlation among items, and weights of items. For example, considering a medical diagnostic database where a record might look like {fever: 80 %, flu: 70 %, AIDS: 30 %, leukemia: 20 %}, whereby the patient is determined to have fever with 80 % probability and so on. Because the diseases such as AIDS and leukemia are less common than fever and flu, they may not appear in the pattern mining results set based on frequency alone. It is therefore necessary to establish a constraint regarding the significance of AIDS and leukemia to ensure that patterns associated with these diseases are found. In addition, the correlation between diseases should also be considered during the process of pattern mining. For instance, a frequent pattern extracted as {fever, AIDS, leukemia} indicates the items are equally correlated, but this is impractical. In fact, {fever, AIDS} or {fever, leukemia} is correlated, but AIDS and leukemia are not. Thus, it is mandatory to establish a constraint on correlation between items when mining patterns. The authors in [3] proposed the expected support, UConf, and wUconf measures to respectively establish the constraints related to frequency, correlation between items, and weights of items when mining WUIPs. Mining WUIPs has practical significance in medical diagnosis, social network behavior prediction, prediction and alarm using IoT sensor systems, etc. Mining WUIPs requires a very fast running time and the capability of extracting interesting patterns that are meaningful to various users.

In this study, we propose an algorithm called HWUIPM that employs a TPN-list structure, an extension of the *N*-list structure [12] for uncertain databases, to improve the performance of the process of mining WUIPs. The main contributions of this manuscript are as follows.

(1) We first introduce the concept of the TPN-list structure, which is used to efficiently store the uncertain database in a linear form so that it is easy to effectively apply divide-and-conquer and pruning strategies to the search space.
(2) We then propose several theorems for the fast calculation and determination of a WUIP based on its TPN-list.
(3) Based on the TPN-list structure and the proposed theorems, we develop the HWUIPM algorithm for mining WUIPs from uncertain databases.
(4) Empirical evaluations are conducted to confirm the effectiveness of the HWUIPM method compared with the state-of-the-art method of mining WUIPs.

The rest of this study is structured as follows. Related works are discussed in Section 2. Section 3 presents the basic concepts associated with mining WUIPs problem. Section 4 introduces the TPN-list structure and the proposed algorithm for mining WUIPs from uncertain databases. Section 5 presents experimental results that demonstrate the effectiveness of the proposed approach. Finally, the conclusion and suggestions for future work are given in Section 6.

## 2. Related works

Many algorithms have been developed to mine frequent itemsets from uncertain databases [2,9,25,26,27,28,13,15,19,20]. In 2007, the U-Apriori algorithm [9] was extended from the Apriori approach for mining frequent patterns from an uncertain database; however, the U-Apriori algorithm has the critical drawback of excessive candidate generation. In 2008, the UF-growth algorithm [26], which used a UF-tree to store the database, was proposed for mining frequent patterns from an uncertain database. In the construction process of the UF-tree [26], frequent 1-itemsets for which the expected support counts were larger than or equal to the minimum support count were first found. The algorithm then removed all the infrequent 1-itemsets from the uncertain database. The construction process of the UF-tree was based on the process of building an FP-tree, with the only difference being that the same items with different existential probabilities were put into different nodes. The UFP-growth algorithm [2] provided a better approach by reducing the search space and employing a pattern-growth paradigm to produce an approximate result. In this way, it was able to avoid generating a large number of candidate-itemsets. To further reduce the size of the UF-tree, UFP-growth groups similar nodes into a cluster, which stores the item $\times$, the maximum existential probability value, and the number of elements. The CUF-growth and CUF-growth* algorithms [27] were later developed to significantly reduce the size of the tree. These approaches outperformed others in terms of mining frequent itemsets from uncertain databases. To reduce the sizes of the UF-tree [26] and UFP-tree [2], a prefix-capped uncertain frequent pattern tree structure called PUF-tree [28] was introduced to compress the information. It was constructed by considering an upper bound on the existential probability value for each item when generating a *k*-itemset. The LUNA algorithm [25] applied a minimum data structure called UP-list to mine uncertain frequent patterns exactly, with no false positives. Several improved techniques have also been developed to allow this algorithm to run more efficiently. Recently, Davashi [15] proposed the UP-tree structure and the UP-Mine algorithm for mining frequent pattern

from uncertain data. The author [15] uses a new tightened upper bound to expected support of patterns which has a significant effect on reducing the number of false positives by tightening the upper bound of expected support and early pruning of infrequent 2-itemsets and their supersets. Kiran et al. [20] proposed a novel model using two tighter upper bound measures to reduce the computational cost for finding all periodic-frequent patterns in an uncertain temporal database. Davashi [13] also introduced two new data structures namely IUP-List and ICUP-List to efficiently store data for incremental mining of uncertain frequent patterns by the ILUNA algorithm. Ovi et al. [19] proposed a novel tree-based algorithm called WFPMUDS-growth for mining weighted frequent pattern from uncertain data streams. The authors [20] used the MWES measure to determine the maximum weighted expected supports of items for prunning the infrequent items.

All of the above approaches are applicable only to the mining of frequent patterns, and do not consider the correlations among the items in frequent itemsets. The number of frequent itemsets is extremely large and carrying out a correlation analysis can remove numerous irrelevant and non-correlated itemsets. This can help the system select the most effective itemsets for knowledge mining. In 2015, Tong et al. [21] proposed the CFP model to exactly represent the distribution of support of an itemset over correlated uncertain data. The authors [21] divided the correlated uncertain database into different groups where transactions in each group are only correlated to each other. They also defined the global frequent itemset that is both frequent in each correlated group and frequent among different groups. Based on the CFP model, Tong et al. [21] designed an Apriori-based algorithm with two pruning and bounding methods to mine correlated frequent itemset and global frequent itemsets efficiently. Recently, Ahmed et al. [3] proposed a new strategy for mining weighted uncertain interesting patterns from uncertain databases in which the patterns could indicate correlations between items with different importance values. The authors [3] proposed the *UConf* and *wUConf* measures for correlation mining from uncertain databases and weighted correlation mining with different weights, respectively. The authors also introduced a prefix proxy value called *pProxy*. This concept improved the frequent pattern mining process in uncertain databases by efficient early pruning of infrequent itemsets.

The *N*-list structure was first proposed by Deng et al. [12] and used to build the PrePost algorithm to represent traditional transaction databases and efficiently find the frequent patterns. The *N*-list structure has subsequently been thoroughly studied and improved in recent studies in frequent patterns mining. Deng et al. [10] proposed the FIN algorithm using the Nodeset structure, a modified version of *N*-list, to mine frequent patterns. FIN made use of a set-enumeration tree to represent the search space, while utilising the superset equivalence property to reduce this search space. Vo et al. [35] proposed the NSFI algorithm, an improved version of PrePost [12], to mine frequent patterns. NSFI used a hash table to speed up the construction of an *N*-list for 1-itemsets and improved the *N*-list intersection operation to ensure correct termination when the threshold is not satisfied. The primary improvement was the application of a "subsume index" to the 1-frequent itemset, which could be used to instantly determine frequent patterns without constructing their *N*-lists. Deng et al. [11] proposed the PrePost + algorithm, which was an improved version of PrePost and employed a pruning technique called children-parent equivalence. With this technique, PrePost + was able to safely prune unpromising branches from the search space. The *N*-list has also been efficiently applied in many other pattern mining tasks, such as the mining of closed frequent patterns [23], frequent weighted patterns [5] and frequent weighted closed patterns [6].

## 3. Basic concepts

For a better understand of the problem, we introduce the concepts of WUIPs mining in Section 3.1. Next, the *N*-list structure, which is the foundation structure of the proposed TPN-list structure to improve the efficiency of WUIPs mining, was summarized in Section 3.2.

### 3.1. WUIP mining

**Definition 1.** *A weighted uncertain database (UDB) is defined as a tuple ⟨T, I, P, W⟩, where* $T = \{t_1, t_2, \cdots, t_m\}, I = \{a_1, a_2, \cdots, a_n\}, P = \{p(a_i, t_j) | i \in [1, n] \wedge j \in [1, m]\}$ *is a set of probabilities for item* $a_i$ *in transaction* $t_j$*, and* $W = \{w_1, w_2, \cdots, w_n\}$ *is a set of corresponding weights of items in I.*

Definition 1 describes the structure of a weighted uncertain database with a specific illustration in Example 1.

**Example 1.** Tables 1 and 2 *show an example UDB with eight transactions. Each transaction contains a number of items which have their own probabilities and weights. The probability of an item may be different for different transactions.*

**Definition 2.** *The expected support of a pattern X in an UDB, denoted as ExpSup(X), is calculated as follows*:

$$ExpSup(X) = \sum_{t \in T_X} \left( \prod_{a \in X} p(a, t) \right) \tag{1}$$

**Table 1**
Example of an uncertain database.

| Transactions | Items | | | | | | |
|---|---|---|---|---|---|---|---|
| $t_1$ | D: 0.6 | A: 0.5 | C: 0.4 | B: 0.9 | E: 0.3 | F: 0.4 | G: 0.3 |
| $t_2$ | D: 0.5 | A: 0.6 | C: 0.4 | B: 0.3 | – | F: 0.5 | G: 0.5 |
| $t_3$ | D: 0.2 | A: 0.7 | C: 0.9 | – | E: 0.4 | – | G: 0.4 |
| $t_4$ | D: 0.2 | A: 0.9 | C: 0.5 | – | E: 0.6 | – | – |
| $t_5$ | D: 0.9 | A: 0.6 | C: 0.6 | B: 0.3 | – | F: 0.4 | G: 0.4 |
| $t_6$ | D: 0.5 | – | – | B: 0.3 | E: 0.6 | F: 0.2 | G: 0.2 |
| $t_7$ | D: 0.4 | – | – | B: 0.2 | E: 0.2 | F: 0.3 | G: 0.1 |
| $t_8$ | D: 0.7 | – | – | B: 0.6 | E: 0.2 | F: 0.2 | – |

**Table 2**
Weights of items.

| Item | D | A | C | B | E | F | G |
|---|---|---|---|---|---|---|---|
| **Weight** | 0.1 | 0.2 | 0.3 | 0.4 | 0.5 | 0.6 | 0.7 |

where $T_X$ is the set of the transactions containing $X$; and $p(a,t)$ is the probability of occurrence of item $a$ in transaction $t$. The *ExpSup* measure is used to determine whether a pattern is frequent or not in an *UDB*.

**Example 2.** *We consider the UDB in Example 1, and calculate the values of ExpSup for the patterns AE and ABE as follows*: $ExpSup(AE) = p(E,t_1) \times p(A,t_1) + p(E,t_3) \times p(A,t_3) + p(E,t_4) \times p(A,t_4) = 0.3 \times 0.5 + 0.4 \times 0.7 + 0.6 \times 0.9 = 0.97$ *and* $ExpSup(ABE) = p(E,t_1) \times p(B,t_1) \times p(A,t_1) = 0.3 \times 0.9 \times 0.5 = 0.135$.

**Definition 3.** [3]. *The expected support confidence of a pattern X in an UDB, denoted as UConf(X), is calculated as follows*:

$$UConf(X) = \frac{ExpSup(X)}{max_{a \in X}(ExpSup(a))} \tag{2}$$

where $a$ is the item contained in the pattern $X$. The *UConf* measure represent the degree of correlation between items in a pattern.

**Example 3.** *For the UDB in Example 1, we have*:

$$UConf(AE) = \frac{ExpSup(AE)}{\max(ExpSup(E), ExpSup(A))} = \frac{0.97}{\max(2.3, 3.3)} = \frac{0.97}{3.3} \approx 0.294$$

**Definition 4.** [3]. *The weighted expected support confidence of a pattern X in an UDB, denoted as wUConf(X), is calculated as follows.*

$$wUConf(X) = \frac{min_{a \in X}(weight(a)) \times ExpSup(X)}{max_{b \in X}(weight(b) \times ExpSup(b))} \tag{3}$$

where $a$ and $b$ are the items contained in the pattern $X$; $weight(a)$ is the weight of item $a$. The *wUConf* measure represent the weight constraint of a pattern in an *UDB*.

**Example 4.** *For the UDB in Example 1, we have*:

$$wUConf(AE) = \frac{min(weight(E), weight(A)) \times ExpSup(AE)}{\max(weight(E) \times ExpSup(E), weight(A) \times ExpSup(A))} = \frac{\min(0.5, 0.2) \times 0.97}{\max(0.5 \times 2.3, 0.2 \times 3.3)} \approx 0.422$$

**Definition 5.** [3] **(***Weighted Uncertain Interesting Pattern* **- *WUIP*): *We consider a UDB and three given thresholds: the minimum expected support threshold $\delta$, the minimum expected support confidence threshold min_UConf, and the minimum weighted expected support confidence threshold min_wUConf. A pattern X in UDB is considered to be a weighted uncertain interesting pattern if* $ExpSup(X) \geq \delta$ and $UConf(X) \geq \text{min\_}UConf$ and $wUConf(X) \geq \text{min\_}wUConf$.

Since mining *WUIPs* directly using Definition 5 involves high complexity and cost, the authors of [6] proposed an approximate mining method with a lower cost. The concepts of $ExpSup^{pCap}$, $uConf^{pCap}$, and $wUConf^{pCap}$ are used to replace *ExpSup*, *Uconf*, and *wUConf*, respectively, in the approximate mining process. $ExpSup^{pCap}$, $uConf^{pCap}$, and $wUConf^{pCap}$ are defined based on the concept of $T^{pCap}$ as follows.

**Definition 6.** [3]**:** *For a UDB, we consider a transaction* $t = \{(a_i, p_i)|i \in [1, n]\}$*, where* $a_i$ *is an item in t,* $p_i$ *is the probability of* $a_i$ *in t, and t is sorted using a predefined order for the items. The transaction prefix cap of transaction t for item* $a_i$*, denoted* $T^{pCap}(t, a_i)$*, is calculated as follows.*

$$T^{pCap}(t, a_i) = \begin{cases} max_{j<i}(p_j) \times p_i if i > 1 \\ \infty if i = 1 \end{cases} \tag{4}$$

**Example 5.** *For the UDB in* Example 1*, according to* Definition 6 *we have*:

$t_1 = \{(D, 0.6), (A, 0.5), (C, 0.4), (B, 0.9), (E, 0.3), (F, 0.4), (G, 0.3)\}$

$T^{pCap}(t_1, \mathrm{D}) = \infty$

$T^{pCap}(t_1, \mathrm{B}) = \max(p_D, p_A, p_C) \times p_B = \max(0.6, 0.5, 0.4) \times 0.9 = 0.54$

**Definition 7.** [3]**.** *Given a UDB, we consider a transaction* $t = \{(a_i, p_i)|i \in [1, n]\}$*, where* $i_k$ *is an item in t,* $p_i$ *is the probability of* $a_i$ *in t, and t is sorted using a predefined order for the items. The transaction prefix proxy of transaction t for item* $a_i$*, denoted* $T^{pProxy}(t, a_i)$*, is calculated as follows*:

$$T^{pProxy}(t, a_i) = \begin{cases} secondmax_{j<i}(p_j) if i \geq 2 \\ 0 if i < 2 \end{cases} \tag{5}$$

**Example 6.** *For the UDB in* Example 1*, according to* Definition 7 *we have*:

$t_1 = \{(D, 0.6), (A, 0.5), (C, 0.4), (B, 0.9), (E, 0.3), (F, 0.4), (G, 0.3)\}$

$T^{pProxy}(t_1, D) = 0$

$T^{pProxy}(t_1, A) = 0$

$T^{pProxy}(t_1, C) = secondmax(0.6, 0.5) = 0.5$

$T^{pProxy}(t_1, B) = secondmax(0.6, 0.5, 0.4) = 0.5$

**Definition 8.** [3]**.** *Given a UDB with transactions sorted using a predefined order for the items and a particular item* $a_i$*. Let* $TP(a_i) = \{TP_k|k \in [1, h]\}$ *be the set of all transactions containing* $a_i$*, where* $TP_k$ *is the set of transactions having the same items in front of* $a_i$ *in their presentations. The prefix proxy of an item* $a_i$*, denoted as* $pProxy_k(a_i)$*, is defined as follows.*

$$pProxy_k(a_i) = max_{t \in TP_k}\left(T^{pProxy}(t, a_i)\right) \tag{6}$$

**Example 7.** *For item E in the UDB in* Example 1*, according to* Definition 8 *we have*:

$TP_1 = \{t_1\}$//with the same items in front of *E* {*D*, *A*, *C*, *B*}.
$TP_2 = \{t_3, t_4\}$// with the same items in front of *E* {*D*, *A*, *C*}.
$TP_3 = \{t_6, t_7, t_8\}$// with the same items in front of *E* {*D*, *B*}.

$pProxy_1(E) = T^{pProxy}(t_1, E) = 0.6$

$pProxy_2(E) = \max(T^{pProxy}(t_3, E), T^{pProxy}(t_4, E)) = max(0.7, 0.5) = 0.7$

$$pProxy_3(E) = \max(T^{pProxy}(t_6, E), T^{pProxy}(t_7, E), T^{pProxy}(t_8, E)) = max(0.3, 0.2, 0.6) = 0.6$$

**Definition 9.** [3]. *Given a UDB with a pattern X = {$a_1$, $a_2$, …, $a_j$}, where $a_u$ $a_v$ ($a_u$ is a proper ancestor of $a_v$) for $1 \le u < v \le j$, the expected support prefix cap of pattern X, denoted as.*

$ExpSup^{pCap}(X)$, is defined as follows:

$$ExpSup^{pCap}(X) = \begin{cases} \sum_{i=1}^{|UDB|} T^{pCap}(t_i, a_j), if\,|X| \le 2 \\ \sum_{i=1}^{|UDB|} \left(T^{pCap}(t_i, a_j) \times pProxy_k(a_j)^{|X|-2}\right) if\,|X| > 2 \end{cases} \quad (7)$$

**Example 8.** *Considering pattern DACB in* Example 1, *according to* Definition 9 *we have*:

$$pProxy_1(B) = 0.6 for t_1, t_2 and t_5$$

$$pProxy_2(B) = 0 for t_6, t_7 and t_8$$

$$ExpSup^{pCap}(DACB) = \sum_{i=1}^{|UDB|} (T^{pCap}(t_i, DACB) \times pProxy_k(DACB)^2) = 0.54 \times 0.36 + 0.18 \times 0.36 + 0.27 \times 0.36 = 0.3564$$

**Lemma 1.** [28]. *If $X_1$ and $X_2$ are two patterns such that $X_1 \subseteq X_2$ and both $X_1$ and $X_2$ have the same suffix item, then:*

$$(i) ExpSup^{pCap}(X_2) \ge \delta \Rightarrow ExpSup^{pCap}(X_1) \ge \delta$$

$$(ii) ExpSup^{pCap}(X_1) < \delta \Rightarrow ExpSup^{pCap}(X_2) < \delta$$

Thus, the formula for calculating $ExpSup^{pCap}$ satisfies the partial downward-closure property.

**Definition 10.** [3]. *Given a UDB, the expected support confidence of pattern X in UDB, denoted as $uConf^{pCap}(X)$, is defined as follows*:

$$UConf^{pCap}(X) = \frac{ExpSup^{pCap}(X)}{max_{a \in X}(ExpSup(a))} \quad (8)$$

**Example 9.** *Considering pattern DACB in* Example 8, *we have*:

$$UConf^{pCap}(DACB) = \frac{ExpSup^{pCap}(DACB)}{max_{a \in DACB}(ExpSup(a))} = \frac{0.3564}{4} = 0.0891$$

**Lemma 2.** [3]. *If $X_1$ and $X_2$ are two patterns such that $X_1 \subseteq X_2$, then:*

$$UConf^{pCap}(X_1) \ge UConf^{pCap}(X_2)$$

Thus, the formula for calculating $uConf^{pCap}$ also satisfies the downward-closure property.

**Definition 11.** [3]. *Given a UDB, the weighted expected support confidence of pattern X in UDB, denoted as $wUConf^{pCap}(X)$, is defined as follows.*

$$wUConf^{pCap}(X) = \frac{min_{a \in X}(weight(a)) \times ExpSup^{pCap}(X)}{max_{b \in X}(weight(b) \times ExpSup(b))} \quad (9)$$

**Example 10.** *For pattern DACB in* Example 8, *we have*:

$$wUConf^{pCap}(DACB) = \frac{min_{a \in DACB}(weight(a)) \times ExpSup^{pCap}(DACB)}{max_{b \in DACB}(weight(b) \times ExpSup(b))} = \frac{0.1 \times 0.3564}{1.04} = 0.034269$$

**Lemma 3.** [3]. *If $X_1$ and $X_2$ are two patterns such that $X_1 \subseteq X_2$,*

$$wUConf^{pCap}(X_1) \geq wUConf^{pCap}(X_2)$$

The measure $wUConf^{pCap}$ fulfils the downward-closure property.

**Theorem 1.** [3]. *Given a UDB and a pattern $X \in UDB$, we have: $ExpSup(X) \leq ExpSup^{pCap}(X)$.*

**Theorem 2.** [3]. *Given a UDB and a pattern $X \in UDB$, we have.: $UConf(X) \leq UConf^{pCap}(X)$*

**Theorem 3.** [3]. *Given a UDB and a pattern $X \in UDB$, we have: $wUConf(X) \leq wUConf^{pCap}(X)$.*

According to Theorems 1–3, the measures $ExpSup^{pCap}$, $UConf^{pCap}$, *and* $wUConf^{pCap}$ are respectively used as upper bounds of *ExpSup*, *Uconf*, and *wUConf* in the WUIP mining process.

**Definition 12.** **(***Mining of approximately WUIPs using the measures* $ExpSup^{pCap}$, $UConf^{pCap}$, *and* $wUConf^{pCap}$**).** *We consider a UDB and three thresholds*: *the minimum expected support threshold* $\delta$, *the minimum expected support confidence threshold min_UConf and the minimum weighted expected support confidence threshold min_wUConf. We can find the approximate set of weighted uncertain interesting patterns in UDB, denoted as WUIP*, using the following formula*:

$$WUIP^* = \{X | (X \in UDB) \wedge (ExpSup^{pCap}(X) \geq \delta) \wedge (UConf^{pCap}(X) \geq \min\_UConf) \wedge (wUConf^{pCap}(X) \geq \min\_wUConf)\}$$

### 3.2. N-list structure

The *N*-list structure was proposed by Deng et al. [12] for mining frequent itemsets on traditional binary databases. The *N*-list structure combines the advantages of FP-tree-based data compression and search space pruning of candidate mining on set-enumerations tree.

First, the transactions in the dataset are preprocessed by removing items that do not satisfy the threshold and reordering the items in descending order of their supports. The transactions are then read and compressed in turn on the PPC-tree, which is an FP-tree-like structure that integrates *pre* and *post* values into nodes for the purpose of determining ancestry relationship between the nodes. Transactions with duplicate items are compressed on the same branch of the PPC-tree. Accordingly, each node in the PPC-tree is a tuple (*item_name*, *child_list*, *pre*, *post*, *count*), where *item_name* is the name of the node created when reading transactions and compressing them on the tree, *child_list* is the list of child nodes of the node, *pre* is the preorder rank of the node when traversing the tree, *post* is the postorder rank of the node when traversing the tree, and *count* represents the number of transactions compressed on that node. The process of building a PPC-tree is illustrated through the following example.

**Example 11.** *We obtain the binary database in Table 3 by removing of the probabilities of the items in the database in Table 1. Given min_support = 70 %, the PPC-tree of the database in Table 3 is built as follows. First, the transactions in the database are read to calculate the support of the items. Items are sorted in descending order of support, and only items with support not less than min_support are kept in the list. Next, the transactions are read for a second time and the items not in the list are removed, the rest are sorted in the order of the list and integrated into the PPC-tree in turn. Each transaction is integrated into the PPC-tree from the root, if the items in the transaction match the existing nodes in the tree, count of that node will be incremented, otherwise a new node will be created with count = 1. Finally, this process traverses the PPC-tree and add pre and post values to the nodes with the preorder rank and postorder rank values respectively. The final PPC-tree is shown in Fig. 1.*

**Table 3**
Binary database.

| **Transactions** | **Items** | | | | |
|---|---|---|---|---|---|
| $t_1$ | D | B | E | F | G |
| $t_2$ | D | B | – | F | G |
| $t_3$ | D | – | E | – | G |
| $t_4$ | D | – | E | – | – |
| $t_5$ | D | B | – | F | G |
| $t_6$ | D | B | E | F | G |
| $t_7$ | D | B | E | F | G |
| $t_8$ | D | B | E | F | – |

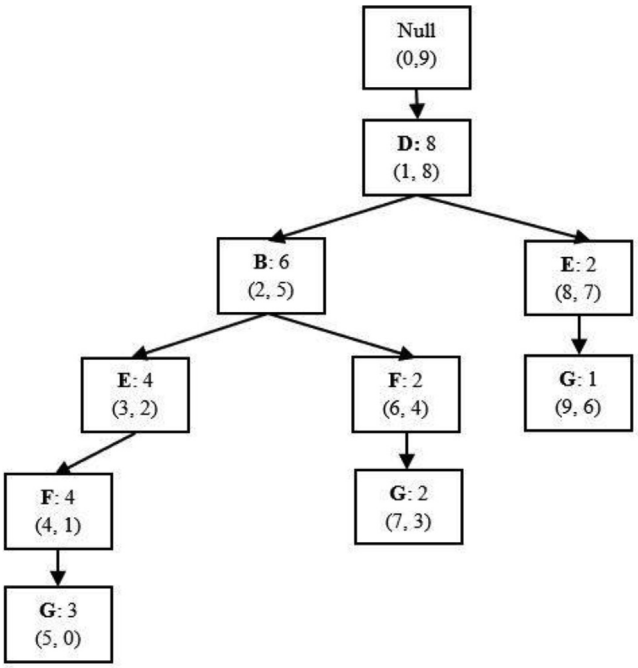


**Fig. 1.** The PPC -tree.

The ancestry relationship between any two nodes can be determined by comparing the *pre* and *post* values of those two nodes. Node $N_1$ is an ancestor of node $N_2$ if and only if $N_1.pre < N_2.pre$ and $N1.post > N2.post$. Example 12 illustrates how to determine the ancestry relationship between two nodes based on their *pre* and *post* values.

**Example 12.** *Given three nodes*: $N_E(3, 2)$, $N_F(4, 1)$ *and* $N_G(7, 3)$, $N_E$ *is an ancestor of* $N_F$ *because* $3 < 4$ *and* $2 > 1$. *Conversely*, $N_E$ *is not the ancestor of* $N_G$ *because it does not satisfy* NE.post > NG.post.

The *N*-list of items is extracted from the PPC-tree and based on the *pre* and *post* values to determine the ancestry relationship between the nodes when performing the *N*-list intersection to find new candidates. The *N*-list of items is extracted from the PPC-tree according to Definition 13 and this process is illustrated in Example 13.

**Definition 13.** [12]: *Given a PPC-tree, let* $N(X) = \{N_1, N_2, \ldots, N_k\}$ *be the set of nodes having item-name = X and these nodes are sorted in ascending order by pre values. N-list of item X, denoted as NL(X), is a sequence of codes as follows* $\{(N_1.pre, N1.post, N_1.count), (N_2.pre, N2.post, N_2.count), \ldots, (N_k.pre, Nk.post, N_k.count)\}$.

**Example 13.** *Considering the PPC-tree in* Fig. 1, *the N-lists of items are as follows*: $NL(D) = \{(1, 8, 8)\}$, $NL(B) = \{(2, 5, 6)\}$, $NL(E) = \{(3, 2, 4), (8, 7, 2)\}$, $NL(F) = \{(4, 1, 4), (6, 4, 2)\}$, $NL(G) = \{(5, 0, 3), (7, 3, 2), (9, 6, 1)\}$.

Mining frequent itemsets is performed based on the *N*-lists of items and is implemented based on a candidate search tree in the form of a set-enumeration tree (Fig. 2). Using a set-enumeration tree makes it easy to implement pruning strategies to reduce search space. The candidates of class $(k + 1)$ are determined based on the intersection of *N*-lists between two corresponding candidates of class *k*. This *N*-list intersection is defined through Definition 14 and is illustrated in detail in Example 14. The support of an itemset *P* with $NL(P) = \{(pre_1, post_1, count_1), (pre_2, post_2, count_2), \ldots, (pre_n, post_n, count_n)\}$ is easily determined by the following formula:

$$support(P) = \frac{\sum_{i=1}^{n} count_i}{m}$$

where *m* is the total number of transactions in the database.

**Definition 14.** [12]**:** *Let PX and PY be two* $(k\text{-}1)$*-itemsets with the same prefix P* (*P can be empty*) *such that item X is behind item Y according to the descending order of support values of items, and NL(PX) and NL(PY) be the two N-lists associated with PX and PY, respectively. The N-list associated with PXY, called NL(PXY), is determined as follows.*

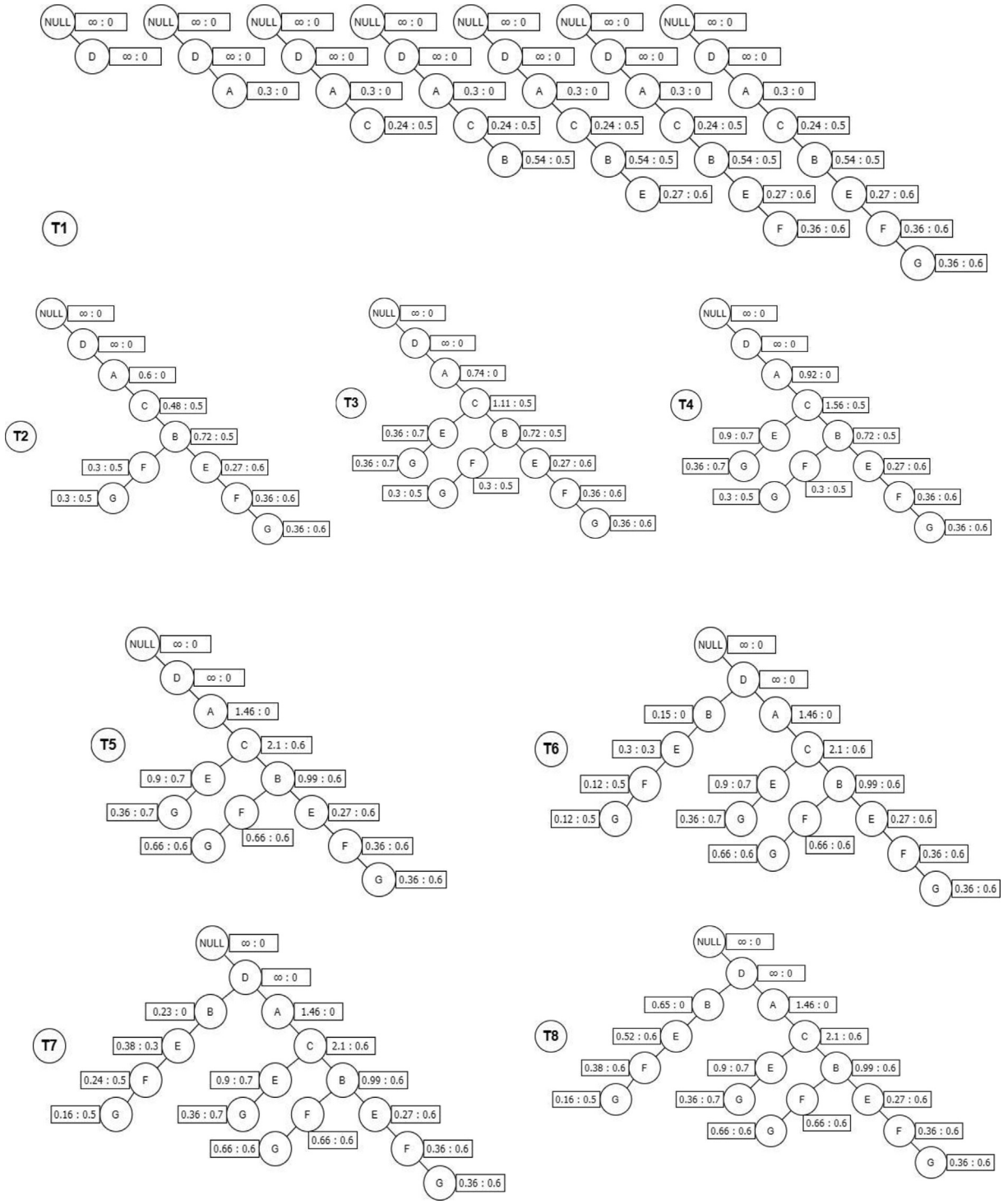


**Fig. 2.** Process of building the example TPN-tree.

1. For each code $C_i(pre_i, post_i, count_i) \in NL(PX)$ and $C_j(pre_j, post_j, count_j) \in NL(PY)$, if $C_j$ is an ancestor of $C_i$ then ($pre_j$, $post_j$, $count_i$) is added to $NL(PXY)$.
2. Traverse $NL(PXY)$ and combine the codes which have the same ($pre$, $post$) into a new code with its *count* value being the total of the *count* values of these codes.

**Example 14.** *According to* Example 13, *we have* $NL(B) = \{(2, 5, 6)\}$ *and* $NL(G) = \{(5, 0, 3), (7, 3, 2), (9, 6, 1)\}$. *Based on* Definition 14, *we calculated* $NL(GB)$ *as follows* $NL(GB) = \{(2, 5, 3), (2, 5, 2)\} = \{(2, 5, 5)\}$.

The *N*-list structure has some advantages as follows when applied to pattern mining problems. Firstly, the information is compressed and encoded to the *N*-lists in a linear form. The mining process can be implemented in the form of the set-enumeration tree and can apply the divide-and-conquer strategy for easy pruning on the search space. Second, the *N*-list intersection has only linear complexity. Finally, the support of an itemset is easily calculated based on the *N*-list of that itemset.

## 4. The proposed method for mining approximately weighted uncertain interesting patterns

In this study, we propose the TPN-tree and the TPN-list structures, based on the *N*-list structure [12], to represent the data and mine *WUIPs*$^*$ from a *UDB* according to Definition 12.

### 4.1. TPN-tree and TPN-list structures

**Definition 15.** **(*TPN-tree*).** *The TPN-tree is a tree structure consisting of a root node and children nodes, in which each child node is a tuple* $\langle item\_name, child\_list, pre, post, T^{pCap}, pProxy\rangle$ *where*:

- *item_name* is the name of the item registered at the node;
- *child_list* is the list of the child nodes of the node;
- *pre* is the order of the node when traversing the tree from top to bottom and left to right;
- *post* is the order of the node when traversing the tree from top to bottom and right to left;
- $T^{pCap}$ records the transaction prefix cap of the node;
- *pProxy* is the prefix proxy of the node.

The TPN-tree is built by scanning preprocessed transactions in the uncertain database one at a time, integrating and compressing the information in the transactions into the tree. Algorithm 1 shows the process of building the TPN-tree from a *UDB* using Definition 15.

**Algorithm 1.**

**Input**: A weighted uncertain database *UDB* and a minimum expected support threshold $\delta$
**Output:** TPN-tree and the frequent expected 1-itemset $I_1$
**Method name:** ***Build_TPN-tree***$(UDB, \delta)$
1. Scan *UDB*and calculate*ExpSup*$(i)$with$\forall i \in I$
I1=i i ∈I ∧ExpSupi≥ δ×|UDB|}Sort I1 in descending order of ExpSup valuesTPN-tree= Nullfor each transaction$t \in UDB$ do
$t = \{i | i \in t \wedge i \in I_1\}$
if t ≠∅ thenSort t based on the order in I1Insert_Tree (t, TPN-tree)end ifend forScan TPN-tree and generate pre and post values for each node**Procedure *Insert_Tree***$(t, Tree)$
Set the current node *CN* as root of *Tree*
for each $i \in t$ do
if $i = CN.child\_list[k].item\_name$ then
$CN.child\_list[k].T^{pCap} += T^{pCap}(t, i)$ *//using equation (6)*
$CN.child\_list[k].p^{Proxy} = \max(T^{pProxy}(t, i), CN.child\_list[k].p^{Proxy})$
else
Create a *newchild* of *CN*
*CN.newchild.item_name* = i
$CN.newchild.T^{pCap} = T^{pCap}(t, i)$ *//using equation (6)*
$CN.newchild.p^{Proxy} = T^{pProxy}(t, i)$ *//using equation (7)*
Set *CN* = *CN.newchild*
end if
end for

**Example 15.** *For the UDB in Example 1 and a minimum expected support threshold $\delta$ = 18 %, the process of building the TPN-tree using Algorithm 1 is illustrated in Fig. 2.*

We first consider the computation and insertion of transaction $T_1$ into the PPC-tree. We in turn read the items in $T_1$ and calculate their $T^{pCap}$ and $T^{pProxy}$ values, and then insert them into the tree (lines 19–23 Algorithm 1). For example, $T^{pCap}(T_1, D) = \infty$ and $T^{pProxy}(T_1, D) = 0$, $T^{pCap}(T_1, A) = 0.3$ and $T^{pProxy}(T_1, A) = 0$. Next, $T_2$ is read and in turn inserted into the tree. In case the nodes of $T_2$ coincide with an existing node on the tree, then add $T^{pCap}$ values to node and compare to get the max value of two $T^{pProxy}$ values (lines 15–17, Algorithm 1). For example, $T^{pCap}(T_2, B) = 0.18$ and $T^{pProxy}(T_2, B) = 0.5$ and the $T^{pCap}$ value of the current node $B$ in the tree is added to 0.18, the $T^{pProxy}$ value remains unchanged. And the process of reading, calculating, and inserting the remaining transactions into the tree continues like that until the end.

After the building the TPN-tree, we traverse the tree to generate the *pre* and *post* values of each node, which are used to determine the ancestor relationships between the nodes of the tree without needing to scan the tree. We obtain the TPN-tree shown in Fig. 3.

**Definition 16.** **(*TPN-code*).** *For a given TPN-tree, the TPN-code of a node N is a tuple* $\langle N.pre, N.pos, N.T^{pCap}, N.pProxy \rangle$.

**Theorem 4.** [12]. *Given two TPN-codes $C_1$ ($x_1$, $y_1$, $t_1$, $p_1$) and $C_2$ ($x_2$, $y_2$, $t_2$, $p_2$), $C_1$ is an ancestor of $C_2$ if and only if $x_1 < x_2$ and $y_1 > y_2$.*

Theorem 4 is used to determine the ancestor–descendant relationship between two TPN-codes without the need to scan the tree to check.

**Example 16.** *Let $C_E$ (8, 5, 0.9, 0.7), $C_G$ (9, 4, 0.36, 0.7) and $C_B$ (10, 11, 0.99, 0.6) be the TPN-codes in Fig. 3. According to Theorem 4, $C_E$ is an ancestor of $C_G$ since 8 < 9 and 5 > 4 but $C_E$ is not an ancestor of $C_B$ as it does not satisfy the condition $C_E.pre > C_B.pre$.*

**Definition 17.** **(*TPN-list of an item*).** *Given a TPN-tree representing a UDB, the TPN-list of an item A in UDB, denoted as TPL(A), is the list of TPN-codes of the nodes registering item X on the TPN-tree, and the TPN-codes in the list are arranged in ascending order of pre values.*

**Example 17.** *The TPN-lists of the items in the TPN-tree in Fig. 3 are as follows.*

$TPL(D) = \{(0, 15, \infty, 0)\}$.
$TPL(A) = \{(6, 13, 1.46, 0)\}$.

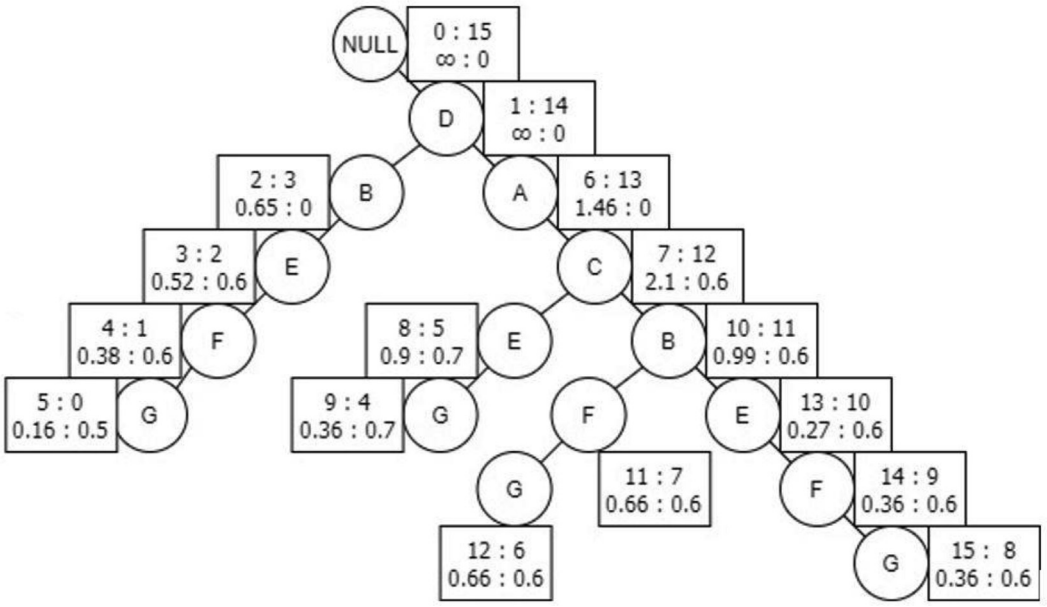


**Fig. 3.** Example TPN-tree with *pre* and *post* values.

*TPL*(*C*) = {(7, 12, 2.1, 0.6)}.
*TPL*(*B*) = {(2, 3, 0.65, 0), (10, 11, 0.99, 0.6)}.
*TPL*(*E*) = {(3, 2, 0.52, 0.6), (8, 5, 0.9, 0.7), (13, 10, 0.27, 0.6)}.
*TPL*(*F*) = {(4, 1, 0.38, 0.6), (11, 7, 0.66, 0.6), (14, 9, 0.36, 0.6)}.
*TPL*(*G*) = {(5, 0, 0.16, 0.5), (9, 4, 0.36, 0.7), (12, 6, 0.66, 0.6), (15, 8, 0.36, 0.6)}.

**Definition 18.** **(*TPN-list of a pattern*).** *Given a UDB, let XA and XB be two* (*k*−1)*-length patterns sharing the same prefix X, where item A is placed behind item B in the order of items in* $I_1$. *TPL*(*XA*) *and TPL*(*XB*) *are two TPN-lists of XA and XB, respectively. The TPN-list of the k-length pattern XBA, denoted as TPL*(*XBA*), *is determined as follows*: (1) *For each pair* $C_i(x_i, y_i, t_i, p_i) \in TPL(XA)$ *and* $C_j(x_j, y_j, t_j, p_j) \in TPL(XB)$, *if* $C_j$ *is an ancestor of* $C_i$ *then we add* $(x_j, y_j, t_i, max(p_i,p_j))$ *into TPL*(*XBA*). (2) *We scan TPL*(*XBA*) *and combine all the TPN-codes with the same pre value into a new TPN-code, where* $T^{pCap}$ *is the sum of all* $T^{pCap}$ *values and pProxy is the maximum value of all pProxy values.*

Based on Definition 18, we can determine the TPN-list of a *k*-length pattern based on the TPN-lists of two corresponding $(k-1)$-length patterns, such as those shown in Algorithm 2.

**Algorithm 2.**

**Input**: Two (k-1)-itemsets *XA, XB* (*B* is before *A* in $I_1$) and their TPN-lists:
*TPL*(*XA*) = $TPL_1$ = {($x_{11}$, $y_{11}$, $t_{11,}$ $p_{11}$), ($x_{12}$, $y_{12}$, $t_{12,}$ $p_{12}$), …,($x_{1m}$, $y_{1m}$, $t_{1m,}$ $p_{1m}$)}
*TPL*(*XB*) = $TPL_2$ = {($x_{21}$, $y_{21}$, $t_{21,}$ $p_{21}$), ($x_{22}$, $y_{22}$, $t_{22,}$ $p_{22}$), …,($x_{2n}$, $y_{2n}$, $t_{2n,}$ $p_{2n}$)}
**Output**: *TPL*(*XAB*)
**Method name**: ***TPL_Intersection***($TPL_1$, $TPL_2$)
1. $TPL_3 = \varnothing, k = 0, i = 1, j = 1$
while j ≤n and (i ≤n) doif TPL2.x2j < TPL1.x1i then if TPL2.x2j > TPL1.x1i thenif k>0 and TPL3.x3k= TPL2.x2j thenTPL3.t3k+= TPL1.t1iTPL3.p3k= max(TPL3.p3k,TPL2.p2j, TPL1.p1i)elsek+=1TPL3←(TPL2.x2j, TPL2.y2j, TPL1.t1i, max(TPL2.p2j,TPL1.p1i)end ifi+=1elsej+=1end ifelsei+=1end ifend whileReturn TPL3

Algorithm 2 calculates the intersection of two TPN-lists and has a complexity of $O(Min(m, n))$, where $m$ and $n$ are the numbers of elements in the two TPN-lists.

**Example 18.** *According to* Example 17, *we have*:
*TPL*(*B*) = {(2, 3, 0.65, 0), (10, 11, 0.99, 0.6)}.
*TPL*(*G*) = {(5, 0, 0.16, 0.5), (9, 4, 0.36, 0.7), (12, 6, 0.66, 0.6), (15, 8, 0.36, 0.6)}.
Applying Algorithm 2, we have:
*TPL*(*BG*) = {(2, 3, 0.16, 0.5), (10, 11, 0.66, 0.6), (10, 11, 0.36, 0.6)}.
= {(2, 3, 0.16, 0.5), (10, 11, 1.02, 0.6)}.

**Theorem 5.** *Given a pattern X with* $TPL(X) = \{(x_1, y_1, t_1, p_1), (x_2, y_2, t_2, p_2), \cdots, (x_n, y_n, t_n, p_n)\}$, *the transaction prefix cap of pattern X can be calculated as follows:*

$$T^{pCap}(X) = \sum_{i=1}^{n} t_i \tag{10}$$

*Proof*: Assume that pattern *X* = {$A_m$, $A_{m-1}$, …, $A_1$}, where $A_i$ is before $A_{i+1}$ in the ordering of $I_1$. According to the TPN-tree building process and Definitions 17–18, we can draw the following conclusions:

(a) For each $C_i(x_i, y_i, t_i, p_i) \in TPL(X)$, there exists a node $N_i$ in the TPN-tree with $N_i.item\_name = A_m$, $N_i.pre = x_i$ and *Ni.post* = $y_i$ corresponding to $(x_i, y_i)$.
(b) $t_i$ is the total of the weight values of all nodes with *item_name* = $A_1$ rooted by node $N_i$ in (a).

Let $T_i$ be the set of transactions containing pattern *X* and traversing the subtree with root $A_m$. Since each $T_i$ corresponds to a TPN-code $C_i(x_i, y_i, t_i, p_i)$ and according to (a), (b) and Equation (4), we have:
$t_i = \sum_{t \in T_i} T^{pCap}(t, A_1)$(c).
If $T(X)$ is the set of all transactions containing pattern *X*, we have:
$T(X) = \bigcup_{i=1}^{n} T_i$(d).
According to (c) and (d), we have:

$$T^{pCap}(X) = \sum_{T_i \in T(X)} \sum_{t \in T_i} T^{pCap}(t, A_1) = \sum_{i=1}^{n} t_i$$

Hence, Theorem 5 is proven.

Using Theorem 5, we can quickly compute the $T^{pCap}$ value for a pattern based on its TPN-list and hence implement an efficient mining process in the search space.

**Example 19.** *According to* Example 18, *we have*:

$TPL(BG)$ = {(2, 3, 0.16, 0.5), (10, 11, 1.02, 0.6)}.

So.$T^{pCap}(BG) = 0.16 + 1.02 = 1.18$

**Theorem 6.** *Given a pattern X with* $TPL(X) = \{(x_1, y_1, t_1, p_1), (x_2, y_2, t_2, p_2), \cdots, (x_n, y_n, t_n, p_n)\}$, *the expected support prefix cap of pattern X can be calculated as follows:*

$$ExpSup^{pCap}(X) = \begin{cases} \sum_{i=1}^{n} t_i if |X| \leq 2 \\ \sum_{i=1}^{n} (t_i \times p_i^{|X|-2}) if |X| > 2 \end{cases} \quad (11)$$

*Proof*: For the case $|X| \leq 2$, we can clearly deduce Equation (11) from Equations (7) and (10). For the case $|X| > 2$, we assume that the pattern $X = \{A_m, A_{m-1}, \ldots, A_1\}$ where $A_i$ is before $A_{i+1}$ in the ordering of $I_1$, and make the same deduction as in the proof of Theorem 5, giving:

$$ExpSup^{pCap}(X) = \sum_{T_i \in T(X)} \sum_{t \in T_i} (T^{pCap}(t, A_1) \times pProxy(A_1)^{|X|-2}) = \sum_{i=1}^{n} (t_i \times p_i^{|X|-2})$$

Hence, Theorem 6 is proven.

Using Theorem 6, we can easily compute the $ExpSup^{pCap}$ value of a pattern based on its TPN-list. The values of $Uconf^{pCap}$ and $wUConf^{pCap}$ can also be calculated quickly, based on Equations (8) and (9), respectively.

**Example 20.** *According to* Example 18, *we have*:

$TPL(BG)$ = {(2, 3, 0.16, 0.5), (10, 11, 1.02, 0.6)}.

Thus,.$ExpSup^{pCap}(BG) = 0.16 + 1.02 = 1.18$

### 4.2. The HWUIPM algorithm

The HWUIPM algorithm for mining $WUIPs^*$ from a *UDB* is given as follows.

**Algorithm 3.**

**Input**: *UDB*, a minimum expected support threshold $\delta$, a minimum expected support confidence threshold *min_uConf*, and a minimum weighted expected support confidence threshold *min_wUConf*.

**Output**: $WUIP^*$.

***Build_TPN-tree***(*UDB*, $\delta$)
Scan *TPN-tree* to generate TPN-lists of items in $I_1$
$WUIP^* \leftarrow I1$***Find_WUIP***($I_1$).
Return $WUIP^*$

**Procedure *Find_WUIP***($I_k$)

for$i \leftarrow |I_k| - 1$down to0 do$I_{next} \leftarrow \varnothing$for$j \leftarrow i - 1$down to0do$TPL_{X_iX_j} \leftarrow$ ***TPL_Intersection***($TPL_{X_i}, TPL_{X_j}$)Calculate$ExpSup^{pCap}(X_iX_j)$//*using Equation (11)*

if($ExpSup^{pCap}(X_iX_j) \geq \delta \times |UDB|$))thenCalculate$uConf^{pCap}(X_iX_j)$//*using Equation (8)*

Calculate$wUConf^{pCap}(X_iX_j)$//*using Equation (9)*

if($uConf^{pCap}(X_iX_j) \geq min\_UConf$)and ($wUConf^{pCap}(X_iX_j)) \geq min\_wUConf$)then $I_{next} \leftarrow X_iX_j WUIP^* \leftarrow X_iX_j$end ifend ifend for***Find_WUIP***($I_{next}$)end for

The operation of the HWUIPM algorithm can be summarised as follows.

- Line #1 constructs the TPN-tree by applying Algorithm 1, as illustrated in Fig. 3.
- Line #2 generates the TPN-list of 1-length patterns in $I_1$ from the TPN-tree.
- Line #3 adds the 1-length patterns in $I_1$ into the result set.
- Line #4 calls the procedure ***Find_WUIP*** to find suitable patterns in the search space.
- Line #4 returns the result set.

- The ***Find_WUIP*** procedure explores the search space based on the set-enumeration tree. It uses the ***TPL_Intersection*** procedure in line #9 to get the TPN-list of a $k$-length pattern from the two TPN-lists of the two corresponding $(k-1)$-length patterns. If the values of $ExpSup^{pCap}$, $UConf^{pCap}$ and $wUConf^{pCap}$ for the returned TPN-list satisfy the thresholds, the new pattern is recorded in the result set. The ***Find_WUIP*** procedure is called recursively, to traverse the search space and generate the candidates by applying a divide-and-conquer strategy.
- Some advantages of the HWUIPM algorithm are as follows: (i) the data are represented linearly in the form of TPN-lists, which is convenient for mining using the set-enumeration tree; (ii) the TPN-list intersection has a linear complexity of $O(n)$; (iii) the TPN-list intersection helps to reduce the size of the TPN-list for a candidate pattern by combining TPN-codes with the same *pre* value; (iv) the value of $ExpSup^{pCap}$ for a pattern can be quickly calculated based on the TPN-list of that pattern using Equation (11).

**Example 21.** *We consider the UDB in Example 1, with the thresholds $\delta = 18\%$, min_Uconf = 10 %, and min_wUConf = 10 %. The TPN-tree is built as shown in Fig. 3. The process of mining WUIPs* is then conducted as shown in Figs. 4–10. First, the search space is initialized by the items in $I_1$ as illustrated in Fig. 4, and the process of generating and checking candidates is then performed step by step, as shown in Figs. 5–10. Finally, the result set is recorded as shown in Fig. 11.*

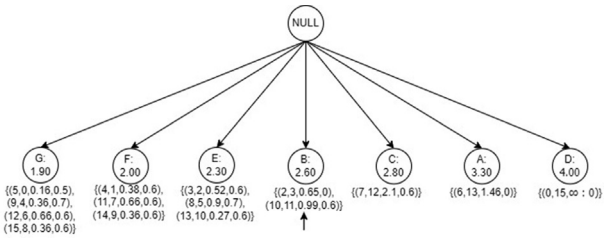


**Fig. 4.** The first layer of the search space.

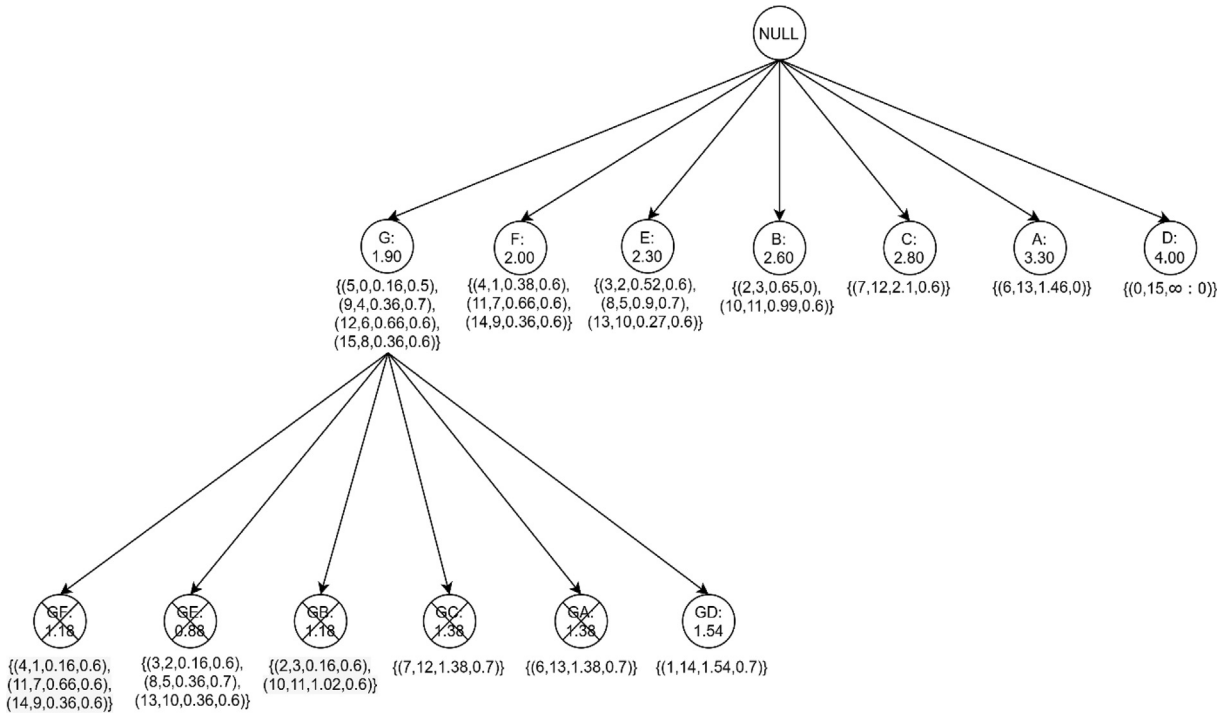


**Fig. 5.** Checking candidates in the equivalence class of *G*.

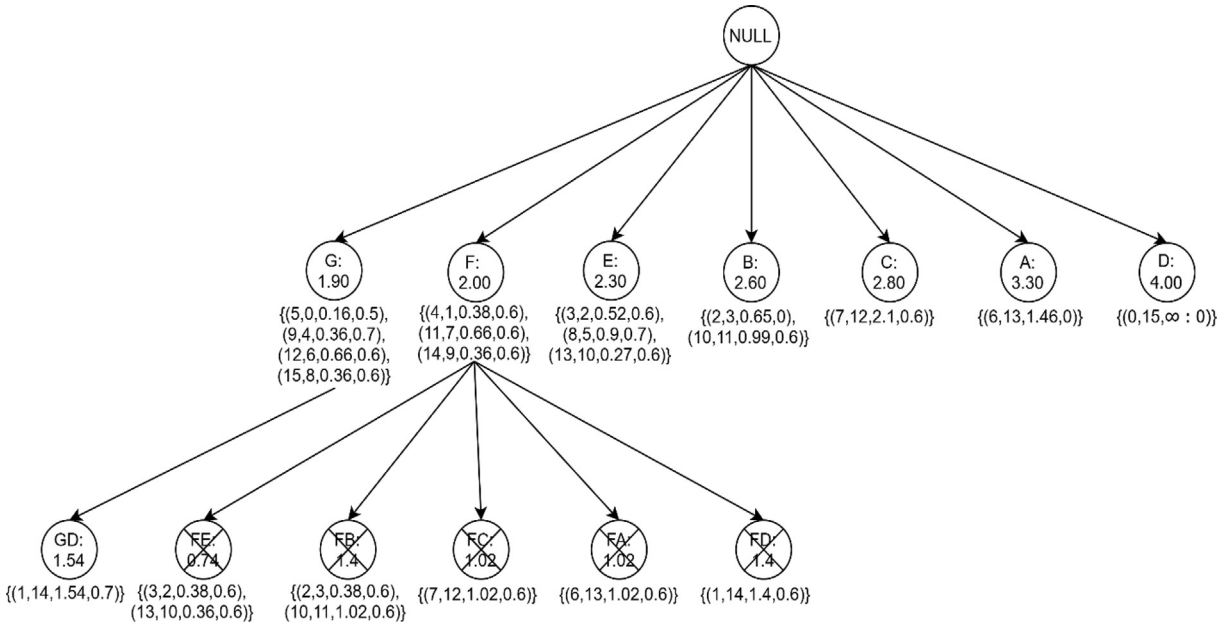


**Fig. 6.** Checking candidates in the equivalence class of *F*.

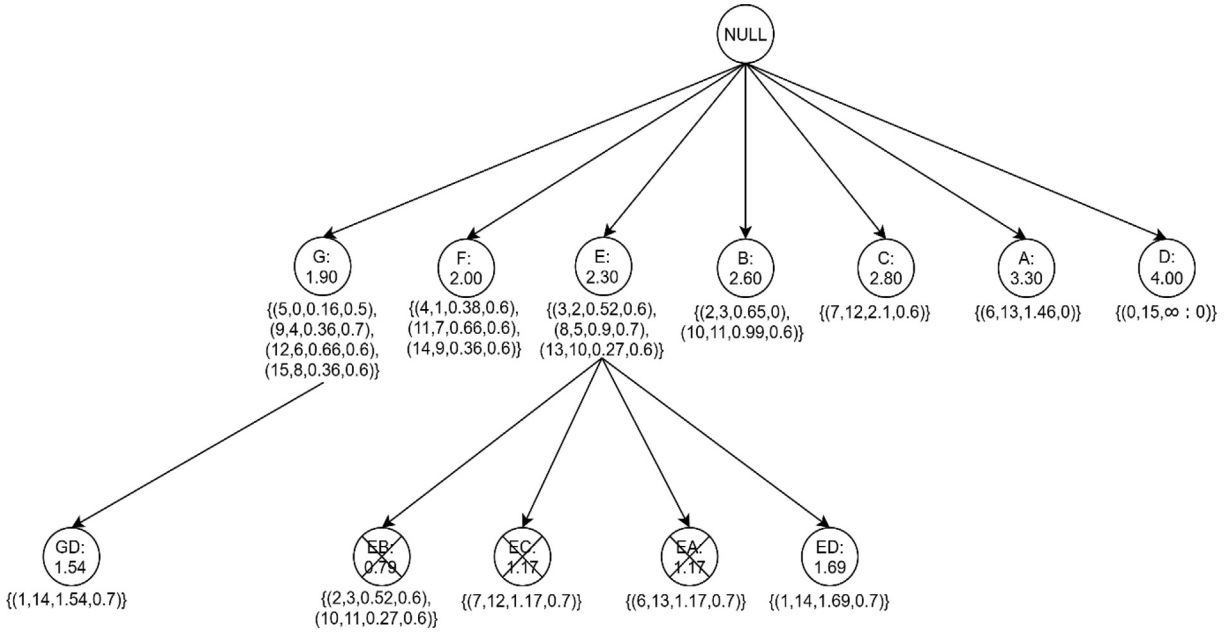


**Fig. 7.** Checking candidates in the equivalence class of *E*.

## 5. Experimental results

All the experiments in this section were performed on a computer with an Intel® Core™ i5-8250 1.8-GHz CPU and 8 GB of RAM. The operating system was Microsoft Windows 10. All algorithms were implemented in C# using the.NET Framework (version 4.5.50709) in Microsoft Visual Studio 2013.

The experimental databases are shown in Table 4; they were obtained from https://fimi.cs.helsinki.fi/data/ and modified by adding randomly generated probability values from 0.1 to 0.9. To demonstrate the effectiveness of the proposed algo-

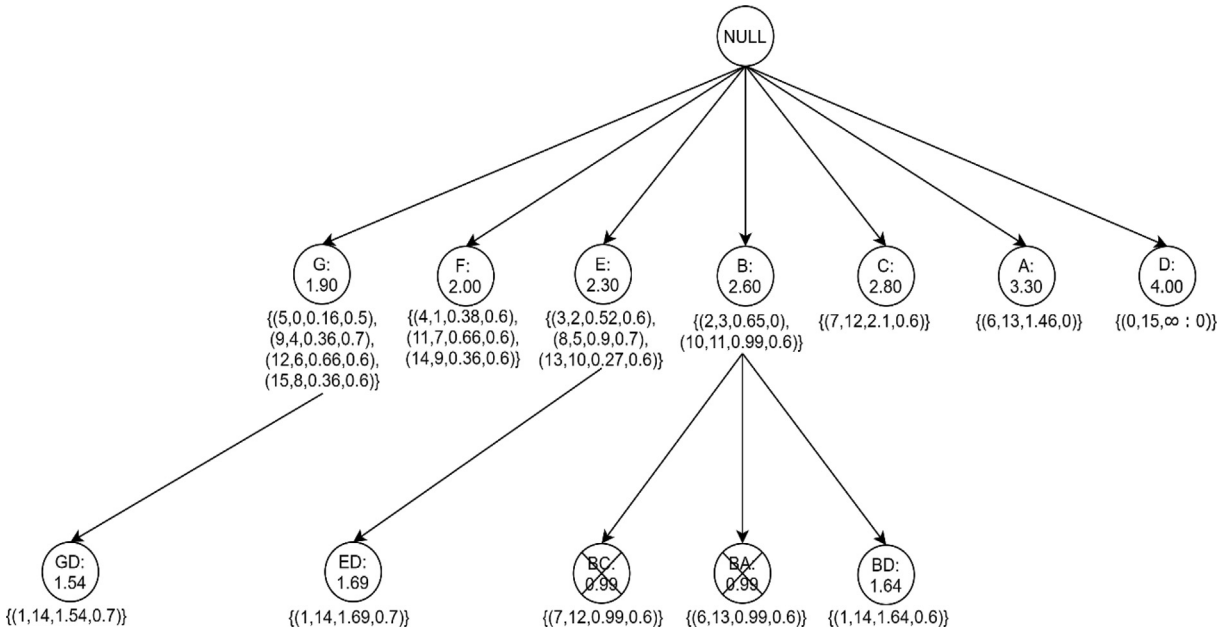


**Fig. 8.** Checking candidates in the equivalence class of *B*.

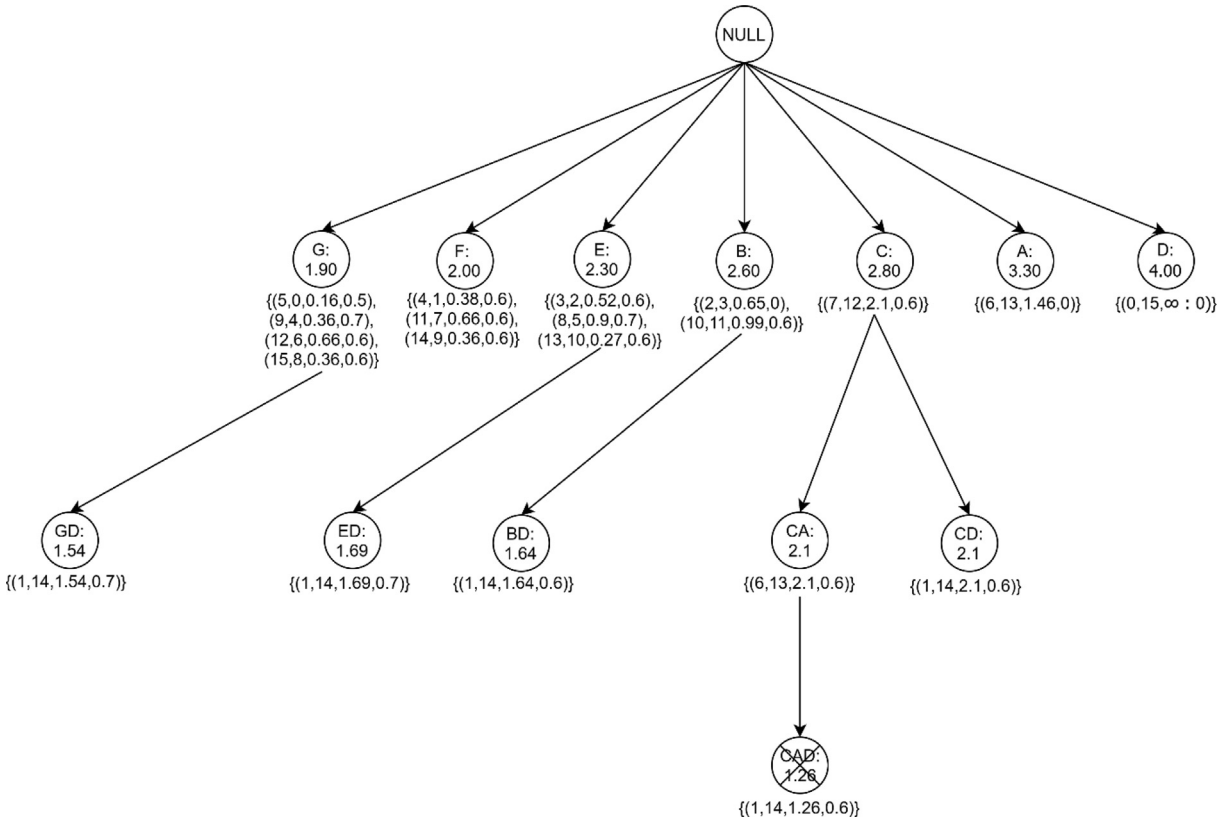


**Fig. 9.** Checking candidates in the equivalence classes of *C* and *CA*.

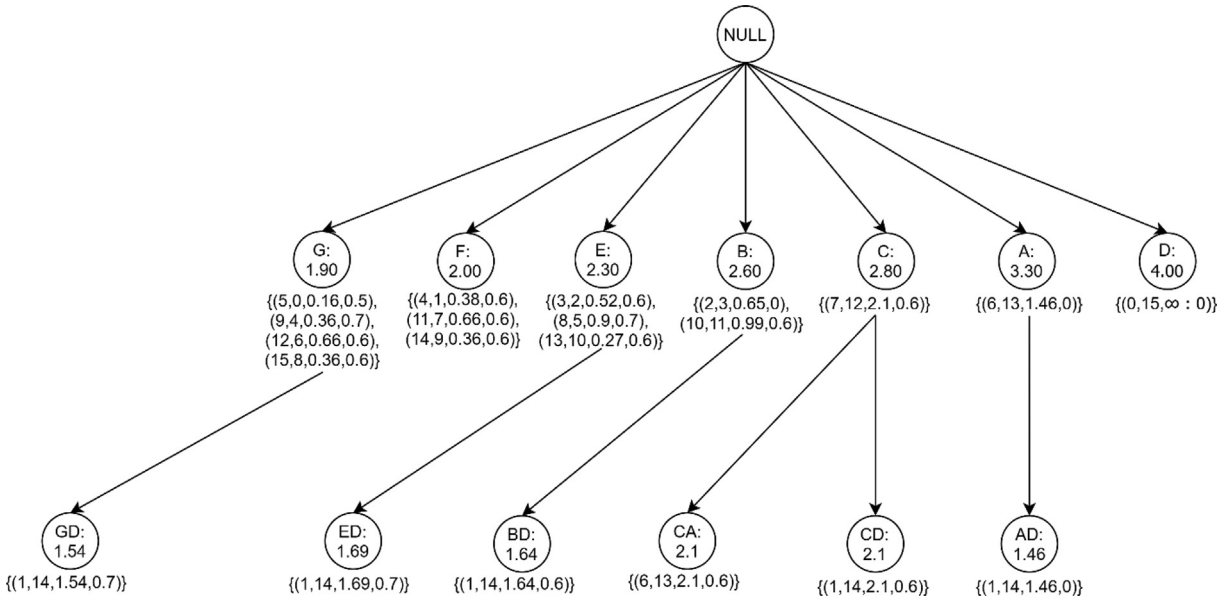


**Fig. 10.** The final result of the mining process.

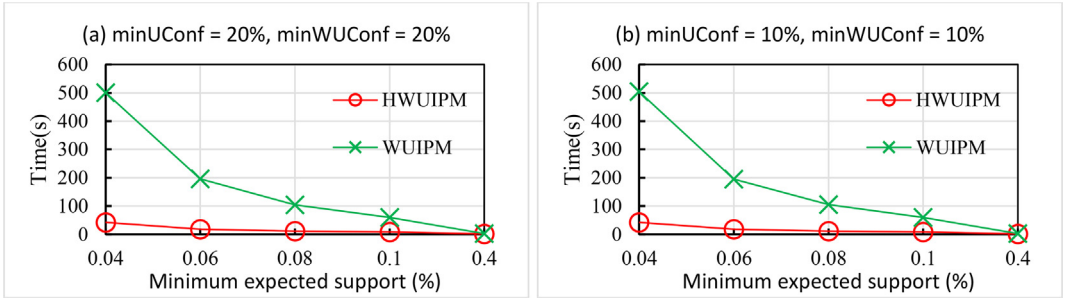


**Fig. 11.** Runtime for the Chainstore database with different *min_UConf* and *min_wUConf* thresholds.

**Table 4**
Features of the experimental databases.

| Database | #Items | #Transactions | Avg. Trans. size |
|---|---|---|---|
| Chainstore | 46,086 | 1,112,949 | 10,3 |
| Kosarak | 41,270 | 990,002 | 7 |
| Retail | 16,470 | 88,162 | 8 |
| BMS-POS | 1657 | 515,597 | 6.5 |
| Connect | 130 | 67,557 | 43 |

rithm compared to the state-of-the-art WUIPM algorithm [3], we conducted experiments in terms of running time, memory usage and scalability on the experimental databases with different *min_UConf* and *min_wUConf* thresholds. The databases used for scalability tests in Table 5 were extracted from the database of Chainstore. We created the databases as Chainstore_300K, Chainstore_500K, Chainstore_700K, Chainstore_900K and Chainstore_1100K by in turn taking first 300000, 500000, 700000, 900000, and 1,100,000 transactions in Chainstore.

### 5.1. Runtime

This section reports the running times for the HWUIPM and WUIPM algorithms for the experimental databases, with different *min_UConf* and *min_wUConf* thresholds. Figs. 11–13 show the running times for the sparse databases, i.e. Chainstore,

**Table 5**
Databases for testing scalability.

| Database | #Items | #Transactions | Avg. Trans. size |
|---|---|---|---|
| Chainstore_300K | 46,086 | 300,000 | 7 |
| Chainstore_500K | 46,086 | 500,000 | 7 |
| Chainstore_700K | 46,086 | 700,000 | 7 |
| Chainstore_900K | 46,086 | 900,000 | 7 |
| Chainstore_1100K | 46,086 | 1,100,000 | 7 |

Kosarak and Retail. For these databases, the performance of HWUIPM and WUIPM was almost equal for a large minimum expected support threshold. However, for smaller minimum expected support thresholds, HWUIPM was significantly more efficient than WUIPM in terms of the mining time. Most of the runtime required by HWUIPM was used to build the TPN-tree, and the mining time was quite low. This is the reason why HWUIPM and WUIPM are almost equally efficient for the large minimum expected support thresholds.

For denser databases such as BMS-POS and Connect (Figs. 14–15), the running times for HWUIPM were consistently lower than for WUIPM, for all minimum expected support thresholds (1, 0.7, 0.4, 0.2 and 0.1 for BMS-POS, and 20, 16, 14, 12 and 10 for Connect). From the experimental results, we can conclude that in terms of the running time, HWUIPM is better than WUIPM for all experimental databases and all parameter values.

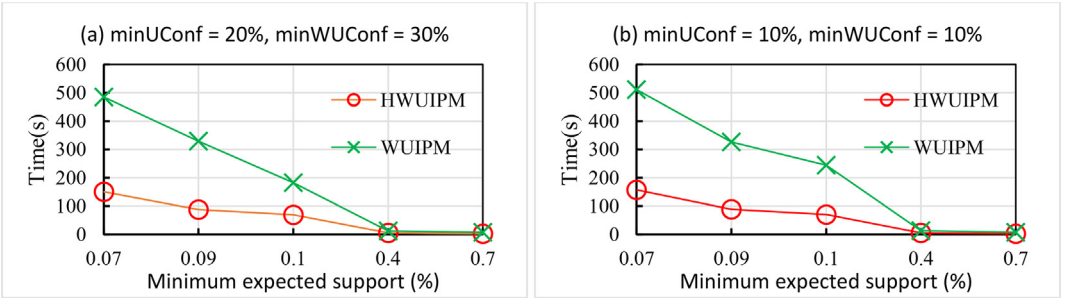


**Fig. 12.** Runtime for the Kosarak database with different *min_UConf* and *min_wUConf* thresholds.

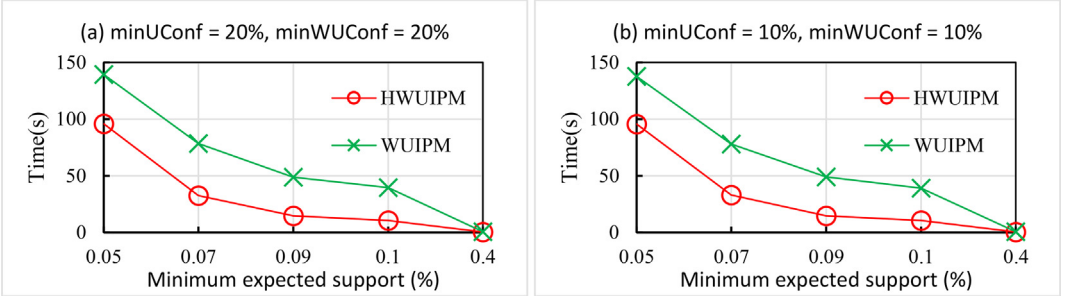


**Fig. 13.** Runtime for the Retail database with different *min_UConf* and *min_wUConf* thresholds.

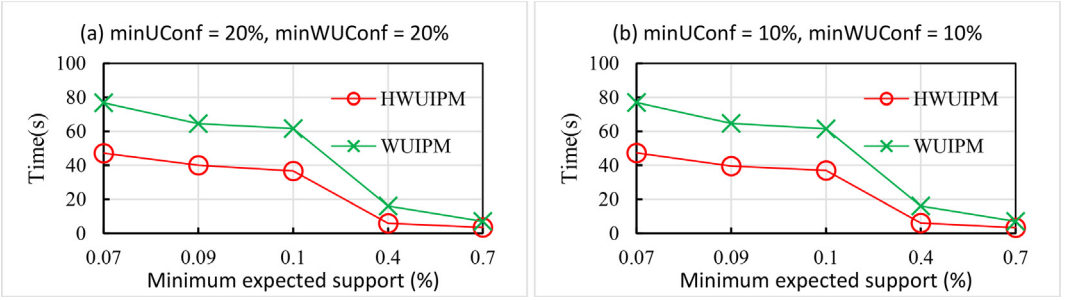


**Fig. 14.** Runtime for the BMS-POS database with different *min_UConf* and *min_wUConf* thresholds.

### 5.2. Memory usage

In this section, we report our results for the memory usage of the HWUIPM and WUIPM algorithms for the experimental databases with different *min_UConf* and *min_wUConf* thresholds.

Fig. 16 shows a comparison between HWUIPM and WUIPM in terms of memory usage for the Chainstore database. It can be seen that HWUIPM is markedly better than WUIPM. For the Kosarak and Retail databases, the difference between the two algorithms is negligible (Figs. 17– 18). HWUIPM is slightly better than WUIPM for a small threshold, whereas for a large threshold, the two algorithms are similar.

For denser databases such as BMS-POS and Connect (Figs. 19–20), the memory usage of HWUIPM was consistently lower than for WUIPM, for all minimum expected support thresholds. This demonstrates that the use of the TPN-list is more effectively than the alternative state-of-the-art structure for mining *WUIPs** for all experimental databases.

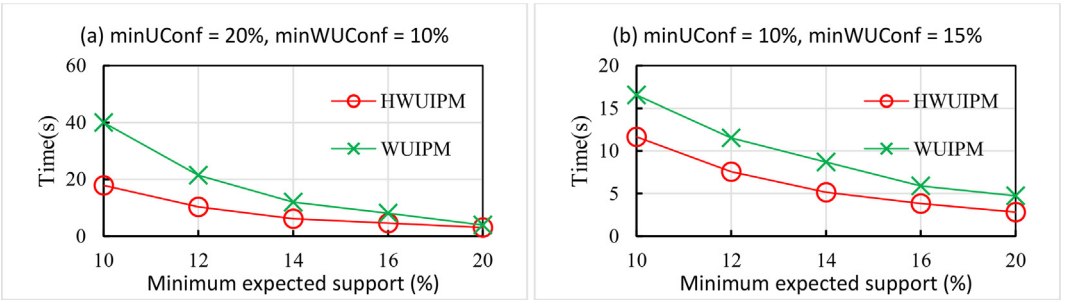


**Fig. 15.** Runtime for the Connect database with different *min_UConf* and *min_wUConf* thresholds.

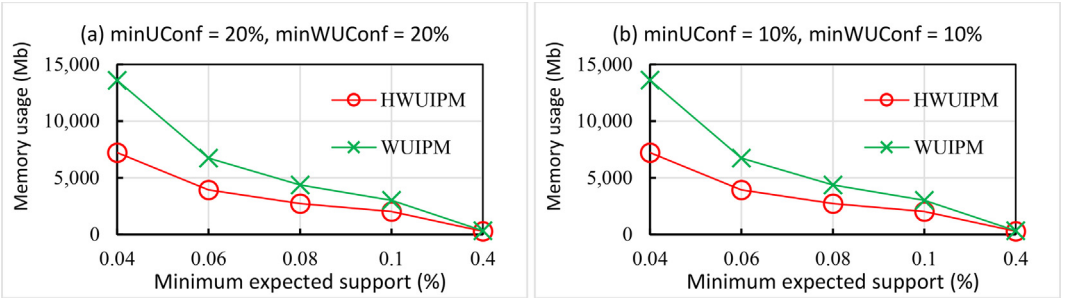


**Fig. 16.** Memory usage for the Chainstore database with different *min_UConf* and *min_wUConf* thresholds.

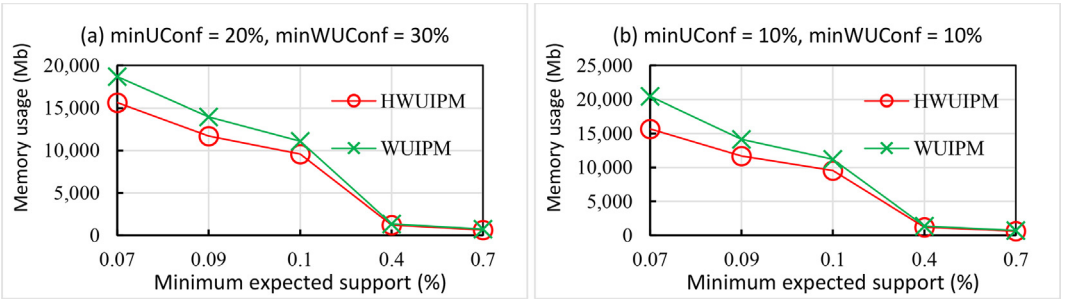


**Fig. 17.** Memory usage for the Kosarak database with different *min_UConf* and *min_wUConf* thresholds.

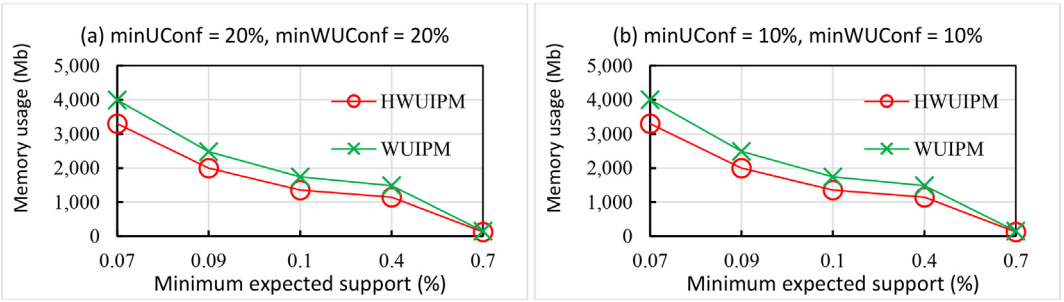


**Fig. 18.** Memory usage for the Retail database with different *min_UConf* and *min_wUConf* thresholds.

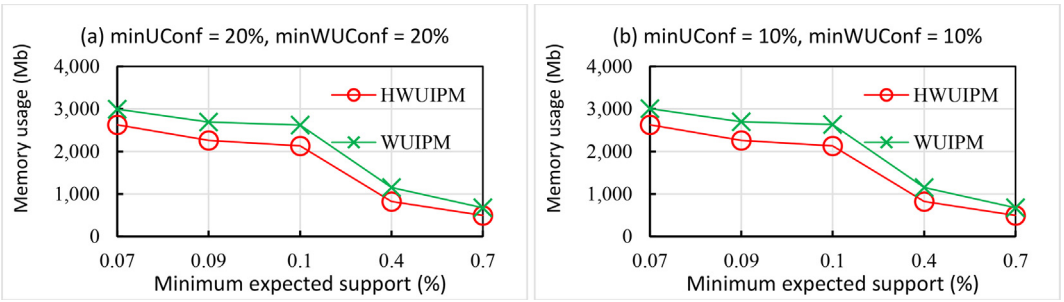


**Fig. 19.** Memory usage for the BMS-POS database with different *min_UConf* and *min_wUConf* thresholds.

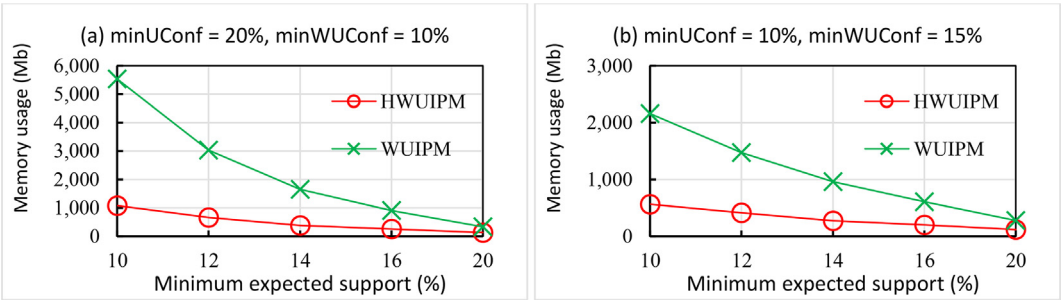


**Fig. 20.** Memory usage for the Connect database with different *min_UConf* and *min_wUConf* thresholds.

### 5.3. Scalability analysis

In this experiment, we set *min_UConf* = 10 % and *min_wUConf* = 10 %, and used four minimum expected support thresholds (0.09 %, 0.08 %, 0.07 % and 0.06 %). We used databases in Table 5 to investigate the running time (Fig. 21) and memory usage (Fig. 22).

From Fig. 21, it can be seen that when the data size increases from 700,000 to 1,100,000 rows, the mining time of WUIPM increases quickly for all four minimum expected support thresholds. In contrast, the running time of HWUIPM remains stable, and is much lower than for WUIPM. Fig. 22 shows the memory usage of these two experimental methods for the above settings. From the results, it can be observed that HWUIPM outperforms WUIPM in terms of memory usage for all data sizes. We can therefore affirm that HWUIPM has better scalability than WUIPM in terms of both running time and resource usage.

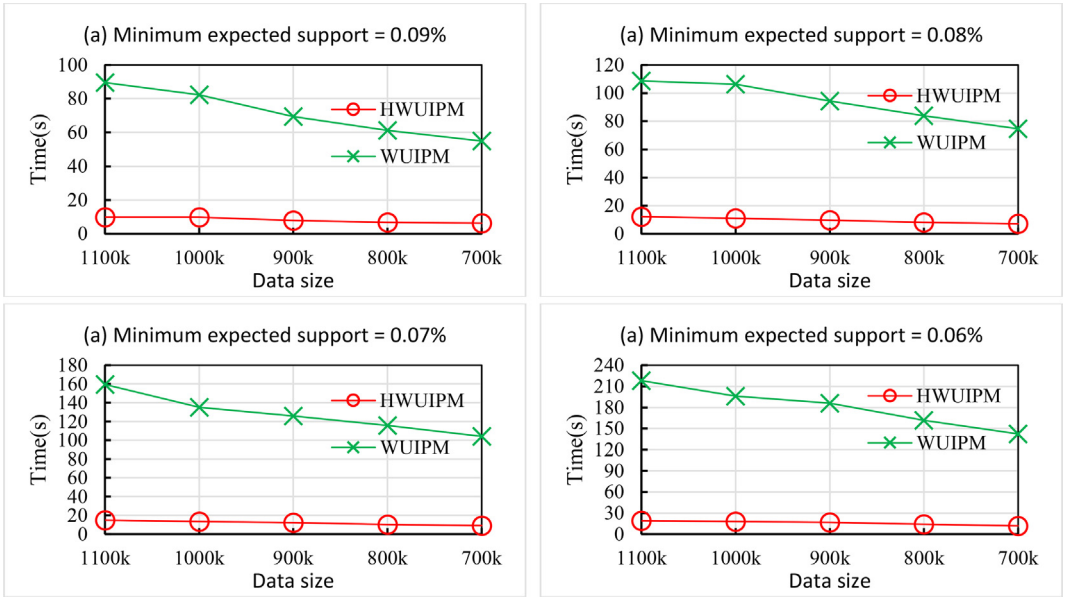


**Fig. 21.** Scalability of runtime (*min_UConf* = 10 % and *min_wUConf* = 10 %) with different minimum expected supports for different data sizes of the Chainstore database.

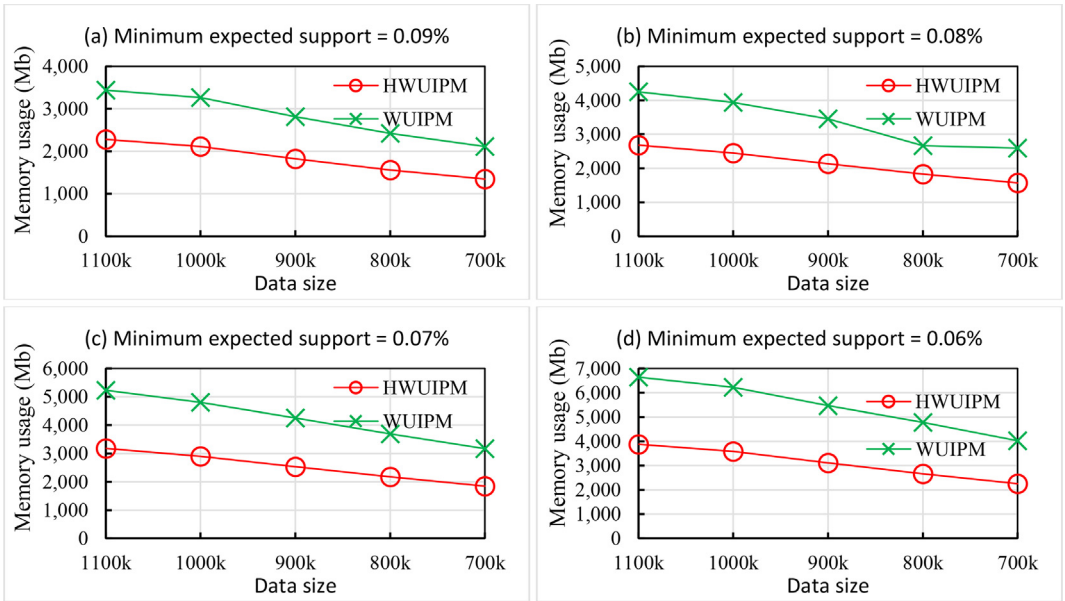


**Fig. 22.** Scalability of memory usage (*min_UConf* = 10 % and *min_wUConf* = 10 %) with different minimum expected supports for different data sizes of the Chainstore database.

## 6. Conclusion

This paper has proposed the HWUIPM algorithm for mining WUIPs from uncertain databases. We first introduced the TPN-list structure, an extended version of the *N*-list structure, to efficiently mine WUIPs, and then presented several theorems for fast calculation and determination of an WUIP based on its TPN-list. The TPN-list structure has many advantages such as: (i) Compressing data on the tree and encoding the ancestry relationship between nodes through *pre* and *post* values; (ii) Converting the data to a linear representation as a list of TPN-lists of items; (iii) Searching the candidates on the set-enumeration tree to apply the divide-and-conquer strategy for easy pruning the search space; (iv) The TPN-list intersection has linear complexity to speed up the computation; (v) The measures $ExpSup^{Cap}$, $UConf^{Cap}$ and $wUConf^{Cap}$ are easily calculated

using TPN-list based on proposed theorems. Using the TPN-list structure and its properties, the proposed HWUIPM algorithm for mining WUIPs from uncertain databases was developed. Finally, we conducted experiments on the running time and memory usage and carried out a scalability analysis. Our experimental results show that the proposed algorithm outperformed the state-of-the-art WUIPM algorithm for mining WUIPs from uncertain databases.

In future work, we will focus on the mining of WUIPs with constraints, such as runtime and numbers of patterns, as well as mining WUIPs from distributed databases. In addition, the parallel approaches for mining WUIPs will be studied to improve the performance in terms of the mining time.

## CRediT authorship contribution statement

**Ham Nguyen:** Conceptualization, Methodology, Writing – original draft. **Dang Vo:** Conceptualization, Software, Writing – original draft. **Huong Bui:** Writing – original draft. **Tuong Le:** Methodology, Validation, Writing – review & editing. **Bay Vo:** Conceptualization, Writing – review & editing, Supervision.

## Data availability

Data will be made available on request.

## 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] Aggarwal C. C., Han J.: Frequent pattern mining, Springer Cham, ISBN978-3-319-07820-5, 2014.
[2] Aggarwal C. C., Li Y., Wang J., Wang J.: Frequent pattern mining with uncertain data. In The 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Paris, France, pp. 29-38, 2009.
[3] U. Ahmed, C.F. Ahmed, M. Samiullah, N. Adnan, C.K. Leung, Mining interesting patterns from uncertain databases, Inf. Sci. 354 (2016) 60–85.
[4] Antonello F., Baraldi P., Shokry A., Zio E., Gentile U., Serio L.: A novel association rule mining method for the identification of rare functional dependencies in Complex Technical Infrastructures from alarm data. Expert Systems with Applications, vol. 170, pid. 114560, 2021.
[5] H. Bui, B. Vo, H. Nguyen, T.A. Nguyen-Hoang, T.P. Hong, A weighted N-list-based method for mining frequent weighted itemsets, Expert Syst. Appl. 96 (2018) 388–405.
[6] H. Bui, B. Vo, T.A. Nguyen-Hoang, U. Yun, Mining frequent weighted closed itemsets using the WN-list structure and an early pruning strategy, Applied Intelligence 51 (2020) 1439–1459.
[7] N. Bui, B. Vo, V.-N. Huynh, C.-W. Lin, L.T.T. Nguyen, Mining closed high utility itemsets in uncertain databases. In Proceedings of the Seventh Symposium on Information and Communication Technology, Ho Chi Minh City, Vietnam, pp. 7-14, 2016.
[8] C.M. Chen, L. Chen, W. Gan, L. Qiu, W. Ding, Discovering high utility-occupancy patterns from uncertain data, Inf. Sci. 546 (2021) 1208–1229.
[9] C.K. Chui, B. Kao, E. Hung, Mining frequent itemsets from uncertain data In PAKDD'07, pp. 47-58, 2007.
[10] Z. Deng, S. Lv, Fast mining frequent itemsets using Nodesets, Expert Syst. Appl. 41 (10) (2014) 4505–4512.
[11] Z. Deng, S. Lv, PrePost+: An efficient N-list-based algorithm for mining frequent itemsets via Children-Parent Equivalence pruning, Expert Syst. Appl. 42 (13) (2015) 5424–5432.
[12] Z. Deng, Z. Wang, J. Jiang, A new algorithm for fast mining frequent itemsets using N-lists, Sci. China Informat. Sci. 55 (9) (2012) 2008–2030.
[13] R. Davashi, ILUNA: Single-pass incremental method for uncertain frequent pattern mining without false positives, Inf. Sci. 564 (2021) 1–26.
[14] Y. Djenouri, M. Comuzzi, Combining Apriori heuristic and bio-inspired algorithms for solving the frequent itemsets mining problem, Inf. Sci. 420 (2017) 1–15.
[15] R. Davashi, UP-tree & UP-Mine: A fast method based on upper bound for frequent pattern mining from uncertain data, Eng. Appl. Artif. Intell. 106 (2021) 104477.
[16] H. Fasihy, M.H.N. Shahraki, Incremental mining maximal frequent patterns from univariate uncertain data, Knowl.-Based Syst. 152 (2018) 40–50.
[17] W. Gan, J.C.W. Lin, P. Fourier-Viger, H.C. Chao, J.M.T. Wu, J. Zhan, Extracting recent weighted-based patterns from uncertain temporal databases, Eng. Appl. Artif. Intell. 61 (2017) 161–172.
[18] W. Gan, J.-C.-W. Lin, P. Fournier-Viger, H.-C. Chao, J. Zhan, Mining of frequent patterns with multiple minimum supports, Eng. Appl. Artif. Intell. 60 (2017) 83–96.
[19] J.A. Ovi, C.F. Ahmed, C.K. Leung A.G., Pazdor Mining weighted frequent patterns from uncertain data streams. In International Conference on Ubiquitous Information Management and Communication (pp. 917-936). Springer, 2019.
[20] Uday Kiran R., Likhitha P., Dao M. S., Zettsu K., Zhang J.: Discovering Periodic-Frequent Patterns in Uncertain Temporal Databases. In International Conference on Neural Information Processing (pp. 710-718). Springer, 2021.
[21] Y.X. Tong, L. Chen, J. She, Mining frequent itemsets in correlated uncertain databases, J. Comput. Sci. Technol. 30 (4) (2015) 696–712.
[22] T. Le, B. Vo, V.-N. Huynh, N.T. Nguyen, S.W. Baik, Mining top-k frequent patterns from uncertain databases, Appl. Intellig. 50 (2020) 1487–1497.
[23] T. Le, B. Vo, An N-list-based algorithm for mining frequent closed patterns, Expert Syst. Appl. 42 (19) (2015) 6648–6657.
[24] G. Lee, U. Yun, K. Ryu, Mining frequent weighted itemsets without storing transaction IDs and generating candidates, Int. J. Uncertainty Fuzziness Knowledge Based Syst. 25 (1) (2017) 111–144.
[25] G. Lee, U. Yun, A new efficient approach for mining uncertain frequent patterns using minimum data structure without false positives, Fut. Generat. Comput. Syst. 68 (2017) 89–110.
[26] C.K.S. Leung M.A.F. Mateo D.A. Brajczuk A tree-based approach for frequent pattern mining from uncertain data. In PAKDD'08, pp. 653-661, 2008.
[27] C.K.S. Leung, S.K. Tanbeer, Fast tree-based mining of frequent itemsets from uncertain data, In DASFAA'12 (2012) 272–287.
[28] C.K.S. Leung, S.K. Tanbeer, PUF-tree: a compact tree structure for frequent pattern mining of uncertain data. In PAKDD'13, pp 13-25, 2013.
[29] J.C.W. Lin, W. Gan, P. Fourier-Viger, T.P. Hong, H.C. Chao, Mining weighted frequent itemsets without candidate generation in uncertain databases, Int. J. Informat. Technol. Decis. Making 16 (6) (2017) 1549–1579.
[30] J.C.W. Lin, W. Gan, P. Fourier-Viger, T.P. Hong, V.S. Tseng, Weighted frequent itemset mining over uncertain databases, Appl. Intellig. 44 (1) (2016) 232–250.

[31] J.-C.-W. Lin, W. Gan, P. Fournier-Viger, T.-P. Hong, V.S. Tseng, Efficient algorithms for mining high-utility itemsets in uncertain databases, Knowl.-Based Syst. 96 (2016) 171–187.
[32] J.-C.-W. Lin, W. Gan, P. Fournier-Viger, T.-P. Hong, V.S. Tseng, Efficiently mining uncertain high-utility itemsets, Soft. Comput. 21 (11) (2017) 2801–2820.
[33] H. Nguyen, B. Vo, M. Nguyen, W. Pedrycz, An efficient algorithm for mining frequent weighted itemsets using interval word segments, Appl. Intellig. 45 (4) (2016) 1008–1020.
[34] M.M. Rahman, C.F. Ahmed, C.-K.-S. Leung, Mining weighted frequent sequences in uncertain databases, Inf. Sci. 479 (2019) 76–100.
[35] B. Vo, T. Le, F. Coenen, T.P. Hong, Mining frequent itemsets using the N-list and subsume concepts, Int. J. Mach. Learn. Cybern. 7 (2) (2016) 253–265.
[36] B. Vo, T. Le, W. Pedrycz, G. Nguyen, S.W. Baik, Mining erasable itemsets with subset and superset itemset constraints, Expert Syst. Appl. 69 (1) (2017) 50–61.
[37] U. Yun, D. Kim, E. Yoon, H. Fujita, Damped window based high average utility pattern mining over data streams, Knowl.-Based Syst. 144 (2018) 188–205.
[38] U. Ahmed, J.C.W. Lin, G. Srivastava, R. Yasin, Y. Djenouri, An evolutionary model to mine high expected utility patterns from uncertain databases, IEEE Trans. Emerg. Top. Computat. Intellig. 5 (1) (2020) 19–28.