Received March 6, 2021, accepted March 22, 2021, date of publication March 31, 2021, date of current version April 19, 2021.

Digital Object Identifier 10.1109/ACCESS.2021.3070132

A Sliding Window-Based Approach for Mining Frequent Weighted Patterns Over Data Streams

HUONG BUI[1,2], TU-ANH NGUYEN-HOANG[1,2], BAY VO[3], (Member, IEEE), HAM NGUYEN[3], AND TUONG LE[4,5]

[1]Faculty of Computer Science, University of Information Technology, Ho Chi Minh City 700000, Vietnam
[2]Vietnam National University, Ho Chi Minh City 700000, Vietnam
[3]Faculty of Information Technology, Ho Chi Minh City University of Technology (HUTECH), Ho Chi Minh City 700000, Vietnam
[4]Informetrics Research Group, Ton Duc Thang University, Ho Chi Minh City 700000, Vietnam
[5]Faculty of Information Technology, Ton Duc Thang University, Ho Chi Minh City 700000, Vietnam

Corresponding author: Ham Nguyen (nd.ham@hutech.edu.vn)

ABSTRACT The mining of frequent weighted patterns (FWPs) that considers the different semantic significance (weight) of items is more suitable for practice than the mining of frequent patterns. Therefore, it plays a vital role in real-world scenarios. However, there exist several limitations when applying methods for mining FWPs designed for static data on growth datasets, especially data streams. Hence, this study proposes an algorithm for mining FWPs over data streams. First, we introduce the concept of mining FWPs over data streams via a sliding window model. Then, we introduce a modification of the weighted node tree (WN-tree) named SWN-tree that has the ability to maintain the information over data streams. Next, this study develops a method for mining FWPs over data streams employing a sliding window model based on SWN-tree. This method is called FWPODS (Frequent Weighted Patterns Over Data Stream) algorithm. Finally, we conduct empirical experiments to compare the performances of our approach and the state-of-the-art algorithm (NFWI) for mining FWPs over data streams. The results of experiment indicate that our approach outperforms the NFWI algorithm when running in batch mode in a sliding window.

INDEX TERMS Pattern mining, data streams, frequent weighted patterns, sliding window model.

I. INTRODUCTION

Mining frequent patterns (FPs) [1], [2] is a topic in artificial intelligence that has attracted much research interest in recent times. Currently, many variations of FPs such as frequent weighted patterns (FWPs) [3]–[9], erasable pattern mining [10]–[13], high utility pattern mining [14]–[17], and high average utility pattern mining [18], [19] have been developed, with many different usage scenarios. In several situations, items in a transaction database can have different importance levels. In mining FWPs, the term "weight" is often used to refer to the importance level of an item. For instance, in retail applications, products with high prices can contribute more to total revenue even though they appear in only a few transactions. In these scenarios, the concept of FWPs [3]–[9] is more suitable for practice than traditional FPs, because it considers the different weights of items. Therefore, it plays a crucial role in such scenarios. Several practical systems, like market data analysis, web traversal pattern mining, and biomedical data analysis, can employ and take advantage of the techniques of mining weight-based patterns. Specifically, each website has a different influence in mining web traversal patterns or some genes that hold special importance in biomedical data analysis to find rare genetic diseases.

In addition, data can be continuously generated at a high-speed in real-life applications. Therefore, discovering patterns over data streams is a great challenge because a massive amount of data cannot be stored in limited memory. Utilizing the methods for mining FWPs over data streams may be impractical due to the limited memory and computing capacity of the system. To overcome this critical drawback of the traditional algorithms, this study develops a cost-effective algorithm for mining FWPs over data streams. The main contributions of this study are as follows. (i) The concept of mining FWPs over data streams via a sliding window model is first introduced in this study. (ii) The SWN-tree for a data stream, an extension of the weighted node tree (WN-tree), is developed. (iii) An FWPODS algorithm based on the WN-list structure generated from SWN-tree is designed

The associate editor coordinating the review of this manuscript and approving it for publication was Fatih Emre Boran.

for mining FWPs over a data stream. (iv) The empirical experiments are conducted to compare the performances of FWPODS and NFWI for mining FWPs over a data stream. The experimental results confirm that FWPODS outperforms the state-of-the-art batch NFWI algorithm when running in batch mode using a sliding-window.

The article is structured as follows. Section 2 surveys related works on mining FWPs in a static dataset, mining patterns over data streams, and N-list-based structures. The FWPODS algorithm is developed in Section 3. Section 4 presents extensive experiments to evaluate and compare the performances of the proposed algorithm and the NFWI algorithm on mining FWPs over data streams. Finally, Section 5 concludes and discusses future work.

II. RELATED WORK

A. MINING FREQUENT WEIGHTED PATTERNS

Initially, Tao *et al.* [3] introduced two concepts: transaction weight and weighted support, and developed an approach to mine FWPs based on two concepts. Specifically, the authors [3] defined the transaction weight as the average of the items' weights in this transaction. Meanwhile, the weighted support of an itemset is determined by the percentage of the total weight of the transactions containing that itemset over the grand total of the weight of all transactions. This model has the advantage of preserving the downward closure property that can help to prune the search space easily. The proposed method [3] utilizes a strategy of creating and checking candidates naively, and therefore it is time-consuming to scan the database many times. Recently, many powerful methods for mining FWPs have been introduced. Vo *et al.* [4] presented the algorithm WIT-FWI-DIFF, which is based on the diffset strategy combined with the WIT-tree data structure, to rapidly mine FWPs. The authors [4] developed some theorems to rapidly calculate weighted supports, giving WIT-FWI-DIFF impressive results in their experiments. After that, Nguyen *et al.* [5] utilized the IWS structure to optimize the tidsets storage space for accelerating the computation of mining FWPs. Based on this structure, the IWS algorithm was introduced. The experimental results confirmed that the IWS algorithm achieved good processing time on sparse datasets. The limitation of IWS is its inefficient performance on dense data such as the Connect dataset. Therefore, Lee *et al.* [6] used two new prefix tree structures FWI-tree^{W} and FWI-tree^{T}, to propose two algorithms FWI*$_{WSD}$ and FWI*$_{TCD}$, respectively, for mining FWPs effectively. Later, using the N-list-based structure, Bui *et al.* [7] proposed the algorithm NFWI for mining FWPs, Vo *et al.* [8] presented TFWIN^{+} for mining top-rank-k FWPs, and Bui *et al.* [9] developed NFWCI for mining frequent weighted closed patterns (FWCPs).

B. MINING PATTERNS OVER DATA STREAMS

In 2002, Manku and Motwani [20] first proposed the problem of mining FPs over data streams. Currently, there are many algorithms, of both precision and approximation types, to mine FPs over data streams, and they can be put into three major groups: landmark-based [21], [22], time-decay based [23], [24], and sliding-window based [25]–[28] models. The landmark-based models only consider data from a landmark which is determined by a specific time point. For instance, the time of starting or restarting a data stream can be set to be a landmark. Yu *et al.* [21] utilized the Chernoff Bound to come close to the FP set in a landmark. Zhi-Jun *et al.* [22] proposed a lattice structure, called a frequent enumeration tree (FET). FET uses equivalence classes (EC) to manage and organize patterns that appear in the same transaction. To avoid storing all patterns in memory, for each EC only the minimum and maximum patterns are retained. In models based on time-decay, the arrival time is used to value the significance of factorial data to focus on recent data. Chang and Lee [23] introduced the estDec algorithm which diminishes the effect of old transactions on the current mining result of a data stream by assigning lower weights to them. After that, Woo and Lee [24] used the damped model to mine frequent maximal patterns (FMPs) over a data stream using the estMax method, which is modified from the estDec algorithm. Finally, many algorithms using the sliding window model were proposed to mine FPs over data streams. Each of them scans data only once at the initial data window to build an initial structure and then effectuates the process of mining patterns based on the built structure over sliding windows. Chiu *et al.* [25] proposed an incremental mining algorithm, DSM-CITI, for discovering frequent inter-transaction patterns from data streams over sliding windows. Deypir and Sadreddini [26], Deypir *et al.* [27] proposed the pWin algorithm based on a sliding window model for mining FPs over data streams with limited memory. pWin retains the information of patterns in a stream using a prefix-tree-based structure that facilitates quick searching of the patterns. Chen *et al.* [28] used the SWP-tree, a summary data structure, and a time decay model for mining FPs in a varying-size sliding window of online data streams. Algorithms based on the sliding window model have also been suggested for the problems of mining other kinds of patterns over data streams, including mining erasable patterns [10]–[12], mining high-utility patterns [14], [15], and mining high-average-utility patterns [18].

C. N-LIST-BASED STRUCTURES

Deng *et al.* [29] first introduced a vertical data structure, the N-list structure, that conserves critical information of frequent itemsets. This structure originates from an FP-tree-like coding prefix tree called PPC-tree. In this study, the authors developed the PrePost algorithm for discovering FPs based on the N-list structure. After that, Deng and Lv [30] presented an improved version of PrePost named PrePost^{+} that uses the Children-Parent Equivalence pruning technique to reduce the candidate search space. Next, Vo *et al.* [1] proposed the NSFI algorithm employing the N-list structure combined with the concept of subsuming to enhance the performance of the mining task. For the problem of mining frequent closed patterns (FCPs), Le and Vo [31] proposed two theorems to quickly check the closed properties of FPs using the N-list structure. Utilizing the two theorems above, the NAFCP algorithm was presented for mining FCPs. Vo *et al.* [32] then developed an approach for the problem of mining FMPs using the N-list structure. Bui *et al.* [7] proposed the WN-list structure, an extended version of the N-list structure, and the NFWI algorithm for the problem of mining FWPs. The results of experiments in this study confirmed that NFWI performs better than the existing approaches for mining FWPs. Later, using the WN-list structure combined with an early pruning strategy, Vo *et al.* [8] and Bui *et al.* [9] proposed TFWIN$^+$ and NFWCI for mining top-rank-k FWPs and FWCPs, respectively.

III. FWPODS: MINING FWPS OVER DATA STREAM

A. SLIDING WINDOW MODEL FOR MINING FWPS OVER DATA STREAM

Let WD be a weighted database. This database contains of a set of transactions $T = \{t_1, t_2, \ldots, t_m\}$. Let $I = \{i_1, i_2, \ldots, i_n\}$ be a set of items, and $W = \{w_1, w_2, \ldots, w_n\}$ be a set of corresponding weights of items in the WD. Each t_i includes a number of items and I is a superset of t_i.

The example weighted database is shown in Tables 1 and 2. This example is used throughout this study. Table 1 describes a dataset including a set of 6 transactions $T = \{t_1, \ldots, t_6\}$. In these transactions, there are 5 items $\{A, B, C, D, E\}$. Table 2 presents the weights of these items.

Definition 1: The transaction weight of $t_k \in T$ denoted by $tw(t_k)$ is determined by the following equation:

$$tw\,(t_k) = \frac{\sum_{i_j \in t_k} w_j}{|t_k|} \tag{1}$$

In other words, $tw(t_k)$ is the average value of all weight values of items belonging to t_k.

Definition 2: The total transaction weight of the weighted database denoted by TTW is the sum total of all transaction weights and is identified as follows.

$$TTW = \sum\nolimits_{t_k \in T} tw(t_k) \tag{2}$$

where T is the set of all transactions in the WD.

Definition 3: Given an itemset X. The weighted support of X denoted by $ws(X)$ is computed by the following equation.

$$ws\,(X) = \frac{\sum_{t_k \in t(X)} tw(t_k)}{TTW} \tag{3}$$

where $t(X)$ is the set of the transactions containing X.

Example 1: From Tables 1 and 2, and Definition 1, we determine the transaction weight of t_2 as follows: $tw(t_2) = \frac{0.5+0.3+0.7}{3} = 0.5$. Similarly, we determine the transaction weight of all remaining transactions in the example weighted database. The results are presented in Table 3. The total transaction weight of the weighted database is 2.98, as shown on the last line in Table 3.

Based on Tables 1 and 3, and Definition 2, we identified the weighted support of itemset BC as follows. Because BC appears in four transactions $\{2, 4, 5, 6\}$, therefore, $ws(BC) = \frac{0.5+0.56+0.42+0.5}{2.98} \approx 0.66$. We have items that have been sorted by descending weighted support in Table 4.

In the sliding-window model, a panel includes several transactions and there are many panels in a window. For instance, consider a window size of 5 transactions and a panel size of 1 transaction for the example WD in Table 1. Figure 1 presents the two windows: window 1 (the left side) and window 2 (the right side). Window 1 consists of the first 5 transactions (T_1 to T_5). Window 2 is acquired by sliding

TABLE 1. An example WD.

TID	Items
1	*A, C, D, F*
2	*A, B, C*
3	*C, D, F, G*
4	*A, B, C, D, G*
5	*A, B, C, D, F*
6	*B, C, E, F*

TABLE 2. The weights of items.

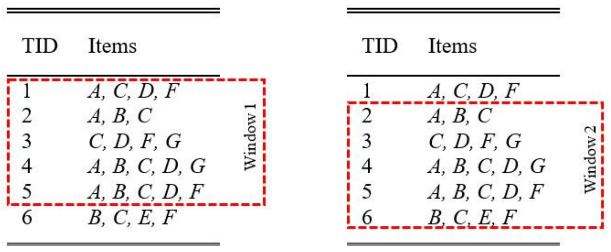
Items	Weights
A	0.5
B	0.3
C	0.7
D	0.4
E	0.8
F	0.2
G	0.9

TABLE 3. The transactional weights in the example weighted dataset.

TID	Transaction weights
1	0.45
2	0.50
3	0.55
4	0.56
5	0.42
6	0.50
TTW	2.98

TABLE 4. The weighted supports of items.

Items	*ws* values
C	1
B	0.664
D	0.664
A	0.648
F	0.644
G	0.372
E	0.168



TID	Items
1	*A, C, D, F*
2	*A, B, C*
3	*C, D, F, G*
4	*A, B, C, D, G*
5	*A, B, C, D, F*
6	*B, C, E, F*

Window 1

TID	Items
1	*A, C, D, F*
2	*A, B, C*
3	*C, D, F, G*
4	*A, B, C, D, G*
5	*A, B, C, D, F*
6	*B, C, E, F*

Window 2

FIGURE 1. The sliding-window model for the example dataset with a window size of 5 and panel size of 1.

window 1 forward by 1 transaction. This step ends by adding T_6 into and removing T_1 from window 1 to create window 2.

The problem examined in this study is mining FWPs over data streams, and thus to find all weighted patterns such that its weighted support satisfies a user-defined minimum weighted support threshold (*min_ws*), i.e. $FWPs = \{X \subseteq I | ws(X) \geq min\_\_ws\}$ in each window.

B. SWN-TREE

In a recent study, Bui *et al.* [7] proposed the WN-list data structure for mining FWPs. First, an algorithm for creating the weighted node tree (denoted by WN-tree) of the weighted database is developed. Each node in this tree consists of five information $\langle name, weight, pre, pos, child\text{-}list \rangle$. In which *name* is the registered item, *weight* is the sum total of *tw* values of the transactions traversing the node, *pre* is the rank order of the node with a pre-order depth-first search, *pos* is the rank order of the node with a post-order depth-first search, and *child-list* is the child nodes of this node. The algorithm generates the WN-list structure from WN-tree. The WN-tree's major weakness is that it cannot update the tree when one or more transactions are removed or inserted as the sliding-window moves forward. Consequently, the WN-tree structure is not appropriate for mining FWPs over data streams. To address this issue, the current study develops the SWN-tree to efficiently mine FWPs over data streams, as follows.

Definition 4: (SWN-tree) An SWN-tree (R), is a tree where nodes contain six elements as follow:

$$\langle name, weight, pre, pos, child - list, parent \rangle \qquad (4)$$

where: (a) *name* is the item identifier, (b) *weight* is the sum total of *tw* values of the transactions containing the item, (c) *pre* is the pre-order rank of the node in depth-first search, (d) *pos* is the post-order rank of the node in depth-first search, (e) *child-list* is the list of child nodes of this node, and (f) *parent* is the parent node.

The proposed SWN-tree construction algorithm (Algorithm 1 - Fig. 2) does not remove unsatisfied items. As a result, the SWN-tree uses more memory storage than the WN-tree. The main superiority of the SWN-tree is that the mining algorithm only generates a tree once, and then this tree can be utilized for mining in various thresholds without recreating the tree. This feature has been created to enable the algorithms to mine FWPs over data streams. Moreover, this study also uses a list of nodes (denoted by TAIL) to retain the last nodes of all products in SWN-tree. TAIL helps to quickly update SWN-tree when one or more transactions are removed.

To illustrate the algorithm building the SWN-tree, the first five transactions in Table 1 are used. In the first step,

Algorithm 1. SWN-tree_Construction
Input: An initial windows *DB*
Output: SWN-tree

```
Generate $R$ as the root of SWN-tree, $R$ = 'null'
For each $T \in DB$ do
    Arrange items in $T$ by frequency in the current window
    Insert_tree($T$, $R$)
Generate pre and pos values for each node in $R$
Return $R$

Procedure Insert_tree($T$, $R$)
While ($T$ is not null) do
    $i \leftarrow$ the first item of $T$
    $T \leftarrow T \setminus i$
    If $N$ is a child of $R$ and $N.name = i$ then
        $N.Weight = N.Weight + tw(T)$
    Else
        Generate a new node $N$
        $N.Weight \leftarrow tw(T)$
        $R.Childnodes = N$
        $N.parent = R$
    If $T$ is null then
        TAIL[$T$] = $N$
    Insert_tree($P$, $N$)
```

FIGURE 2. Construction of the SWN-tree.

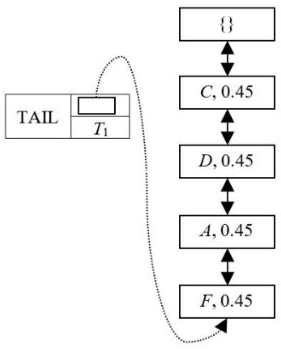


FIGURE 3. SWN-tree for T_1 in the example dataset.

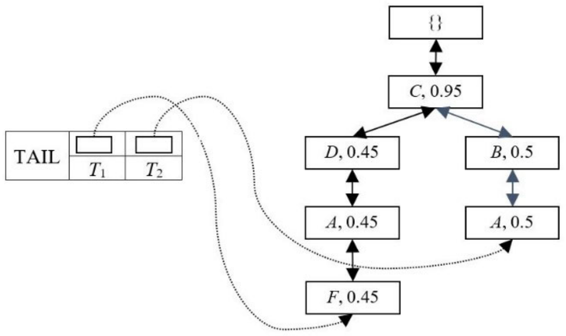


FIGURE 4. SWN-tree for T_1 to T_2 in the example dataset.

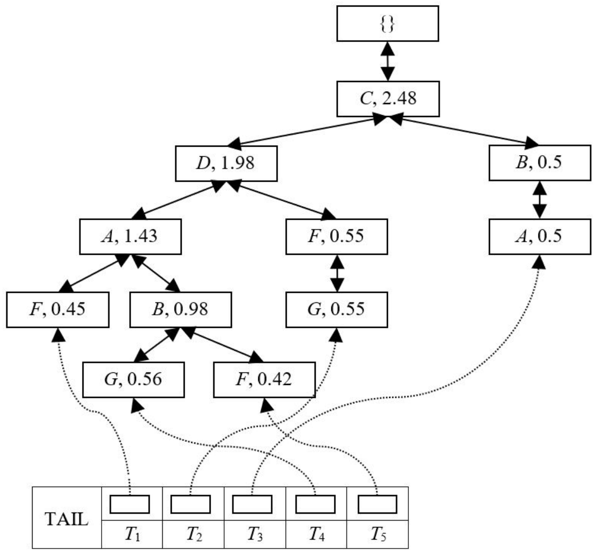


FIGURE 5. SWN-tree for T_1 to T_5 (Window 1) in the example dataset.

Algorithm 2. Maintaining-SWN-tree
Input: SWN-tree R and T – new inserted transactions
Output: Updated SWN-tree for the new window

```
1   For each T_i in T do
2       Call Insert_tree(T_i, R)
3   For i ← 1 to |T| do
4       Let l be the first element in TAIL, N = l.Parent and t = l.TID
        do
5           N.weight = N.weight – tw(t)
6           if N.weight = 0 then
7               Let N' = N.Parent
8               Delete N
9               Let N = N'
10          else
11              N = N.Parent
12      while (N is not the ROOT of R)
13      Delete l in TAIL
14  Return R
```

FIGURE 6. The algorithm for maintaining SWN-tree.

transaction $\{A, C, D, F\}$ with $tw(T_1) = 0.45$ is inserted into the SWN-tree. The sorted transaction of T_1 is $\{C, D, A, F\}$. The result of this step is shown in Fig. 3. Next, the transaction $T_2 = \{A, B, C\}$ is inserted with $tw(T_2) = 0.5$ into the SWN-tree. The sorted transaction of T_2 is $\{C, B, A\}$. The resulting tree of the second step is shown in Fig. 4. After that, each transaction from T_3 to T_5 is added into the tree with the same way. The SWN-tree as determined after adding the first five transactions (Window 1) of the example dataset is presented in Fig. 5.

C. MAINTAINING SWN-TREE OVER DATA STREAM

This section presents the Maintaining-SWN-tree algorithm for maintaining information about FWPs in a data stream using the SWN-tree structure (Algorithm 2 – Fig. 6).

Sliding the window over data streams is performed in two steps. First, the transaction(s) of the current window is inserted into the tree. Then, the last transaction(s) in the previous window is removed from the tree. Figure 7 shows the SWN-tree after Algorithm 2 inserts transaction T6. Then, T1 will be removed from the SWN-tree that is presented in Fig. 8.

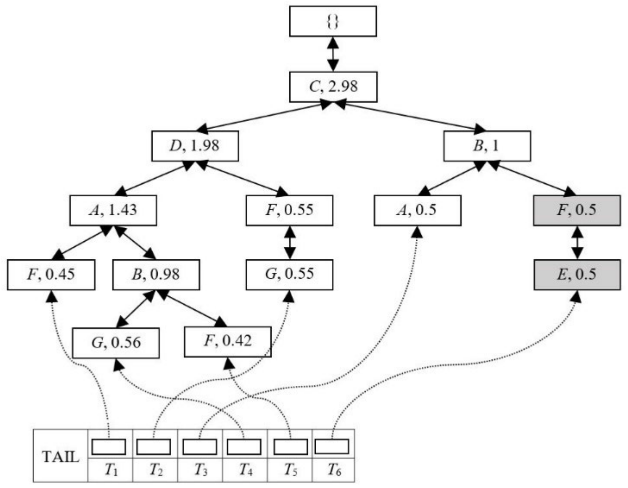


FIGURE 7. SWN-tree for T_1 to T_6 in the example dataset.

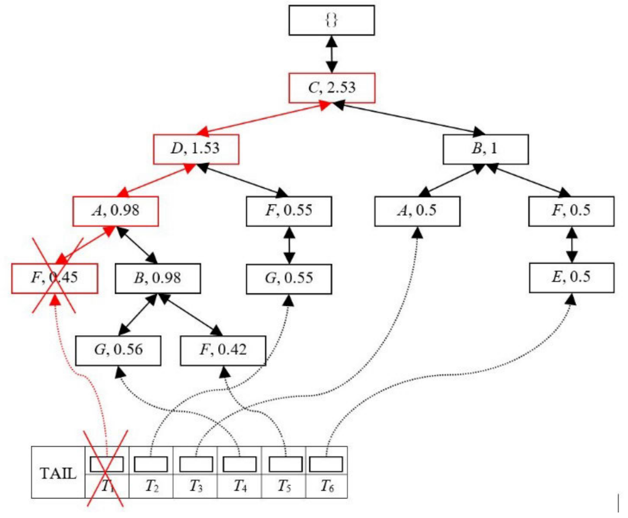


FIGURE 8. SWN-tree for T_2 to T_6 (Window 2) in the example dataset.

TABLE 5. The features of given datasets.

Dataset	# of Transactions	# of Items	Average length	Density	Type
Accidents	340,183	468	33.8	0.072	Sparse
Connect	67,557	130	43	0.333	Dense
Kosarak	990,002	41,270	8	0.0002	Sparse
Pumsb	49,046	7,117	74	0.035	Sparse
Retail	88,162	16,470	10.3	0.0006	Sparse

D. FWPODS ALGORITHM

This section presents the FWPODS algorithm for mining FWPs over data stream (Algorithm 3 – Fig. 9).

Algorithm 3. FWPODS algorithm

Input: SWN-tree of the current window (R), the new inserted transactions (T), and a threshold min_ws
Output: FWPs in the new sliding-window

```
1   R ← Maintaining-SWN-tree(R, T)
2   Scan R to generate $I_1$ and its WN-list which satisfy the $min\_ws$
3   Let FWPs = $I_1$
4   Find_FWP($I_1$)
5   Return FWPs

    Procedure Find_FWP($Is$)
1   For $i = |Is|$ downto 1 do
2       $I_{next} = \varnothing$
3       For $j = i - 1$ to 0 do
4           $WL(X_iX_j)$ = WL_Intersection($WL(X_i)$, $WL(X_j)$)
5           If $ws(X_iX_j) \geq min\_ws$ then
6               $FWPs \leftarrow X_iX_j$
7               $I_{next} \leftarrow X_iX_j$
8       If $I_{next} \neq \varnothing$ then
9           Find_FWP($I_{next}$)

    Procedure WL_Intersection($WL_1$, $WL_2$)
1   $WL_3 = \varnothing$
2   Let $k = 0, i = 1, j = 1$
3   While $(i \leq m)$ and $(j \leq n)$ do
4       If $(WL_2.pre_{2j} < WL_1.pre_{1i})$ then
5           If $(WL_2.pos_{2j} > WL_1.pos_{1i})$ then
6               If $(k > 0)$ and $(WL_2.pre_{2j} = WL_3.pre_{3k})$ then
7                   $WL_3.weight_{3k}$ += $WL_1.weight_{1i}$
8               Else
9                   $k++$
10                  Add ($WL_2.pre_{2j}$, $WL_2.pre_{2j}$, $WL_1.weight_{1i}$) into $WL_3$
11              $i++$
12          Else
13              $j++$
14      Else
15          $i$++
16  Return $WL_3$
```

FIGURE 9. The algorithm for mining FWPs over the data stream.

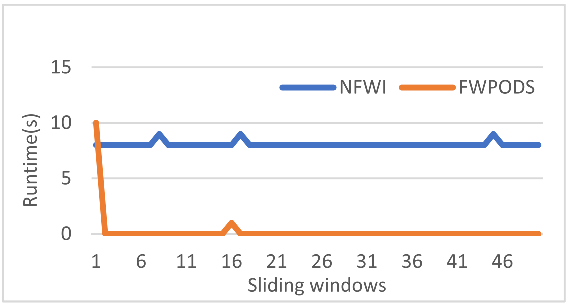


FIGURE 10. Runtime on Accident with *min_ws* = 20%, window_size = 340,000 and panel_size = 1.

The algorithm assumes that we already have the SWN-tree of the current window (R). When the newly inserted transactions are added, the algorithm's task is to provide all FWPs that satisfy the *min_ws* in the new window. Firstly, the FWPODS algorithm calls the Maintaining-SWN-tree algorithm to maintain the SWN-tree (Line 1). Specifically, the Maintaining-SWN-tree algorithm will add the new transaction(s) and remove the last transaction(s) from the SWN-tree. Secondly, FWPODS scans the new SWN-tree at step 1 to generate 1-frequent weighted patterns (1-FWPs) that satisfy the *min_ws* and WN-list structure (Line 2). The *pre* and *pos*

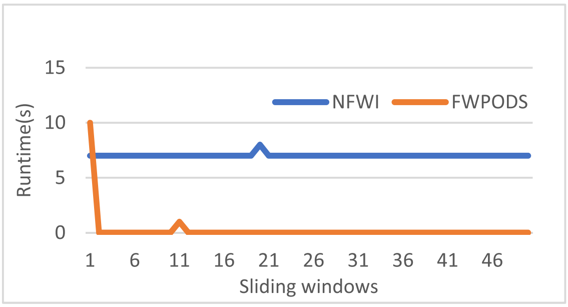


FIGURE 11. Runtime on Accident with *min_ws* = 30%, window_size = 340,000 and panel_size = 1.

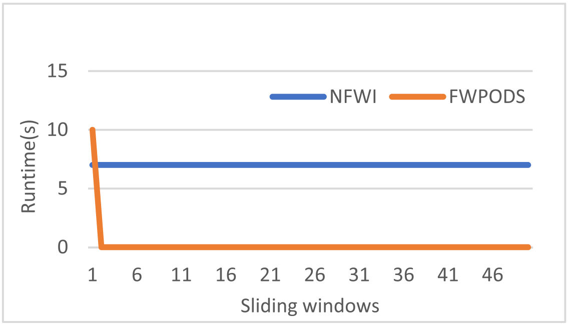


FIGURE 12. Runtime on Accident with *min_ws* = 40%, window_size = 340,000 and panel_size = 1.

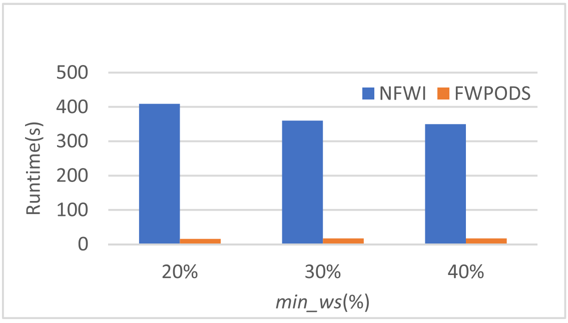


FIGURE 13. Runtimes on Accident in 50 sliding windows with different values of *min_ws*.

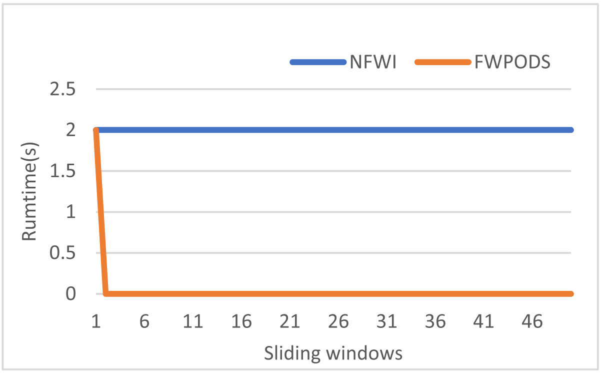


FIGURE 14. Runtime on Connect with *min_ws* =70%, window_size = 67,000 and panel_size = 1.

values of each node on the tree are generated to create WN-lists of 1-FWPs in this step. Next, the algorithm will call the Find_FWP procedure to mine all FWPs which satisfy

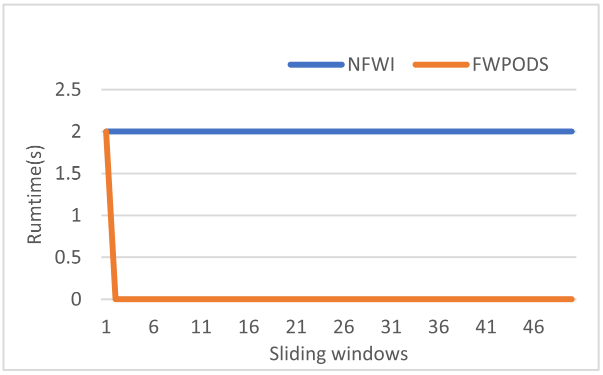


FIGURE 15. **Runtime on Connect with *min_ws* = 80%, window_size = 67,000 and panel_size = 1.**

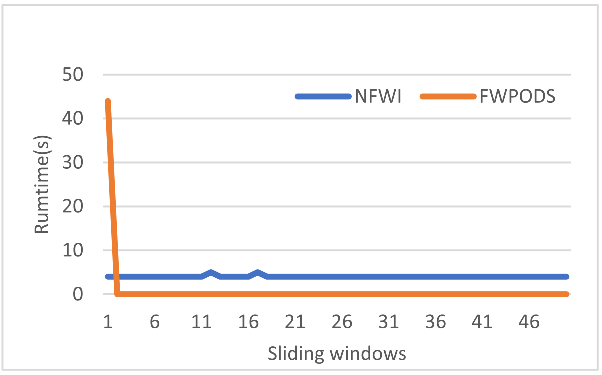


FIGURE 18. **Runtime on Kosarak with *min_ws* = 0.3%, window_size = 500,000 and panel_size = 1.**

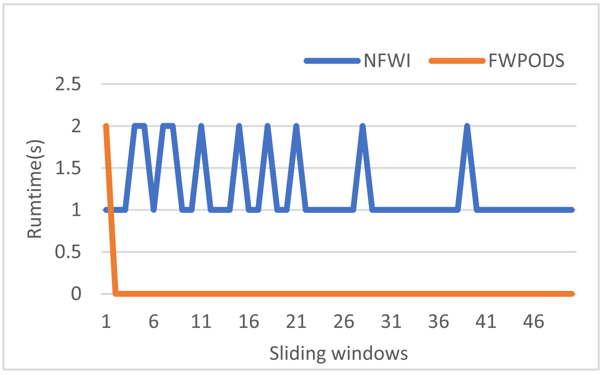


FIGURE 16. **Runtime on Connect with *min_ws* = 90%, window_size = 67,000 and panel_size = 1.**

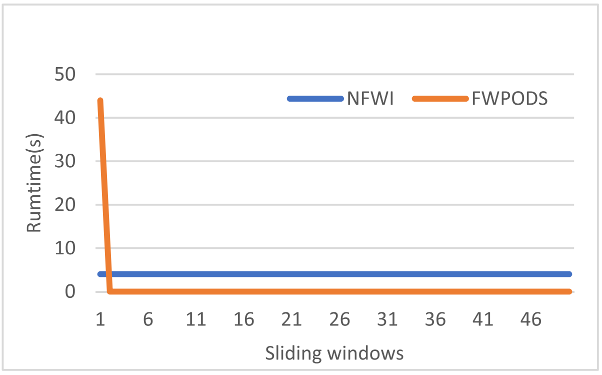


FIGURE 19. **Runtime on Kosarak with *min_ws* = 0.4%, window_size = 500,000 and panel_size = 1.**

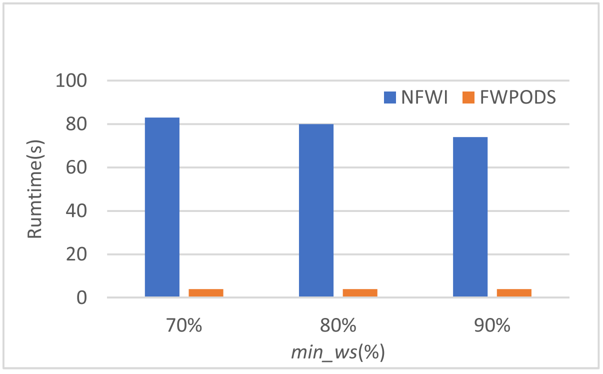


FIGURE 17. **Runtimes on Connect in 50 sliding windows with different values of *min_ws*.**

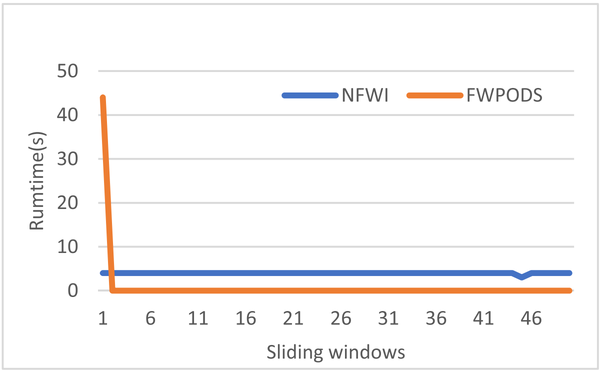


FIGURE 20. **Runtime on Kosarak with *min_ws* = 0.5%, window_size = 500,000 and panel_size = 1.**

the *min_ws* in the new window. The Find_FWP procedure that adopts a divide-and-conquer strategy combined with the WN-list structure presented in detail in Algorithm 3.

IV. EXPERIMENTS

A. EXPERIMENTAL SETTING

The experiments in this study are performed on a laptop equipped with the Windows 10 operating system, an Intel Core i7-7500U 32.70GHz and 32 GB of RAM. The NFWI and FWPODS algorithms are implemented in C# with the. Net Framework (version 4.8.04084). The experiments are performed on five benchmark datasets downloaded from [33]. Table 5 shows the features of the given datasets.

According to the column of density (i.e., density = #Average length of transaction / # of Items), Accidents, Kosarak, Pumsb, and Retail are sparse (density < 0.1) while Connect is dense (density ≥ 0.1). Note that the experimental datasets are modified by setting random values in the range of [1], [10] to be item weights. The modified version of them is provided by Bui *et al.* [7].

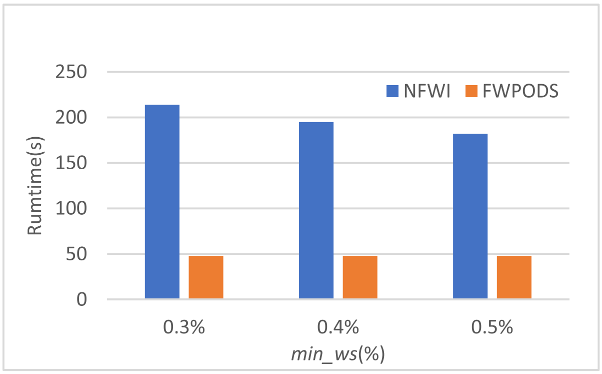


FIGURE 21. Runtimes on Kosarak in 50 sliding windows with different values of ***min_ws***.

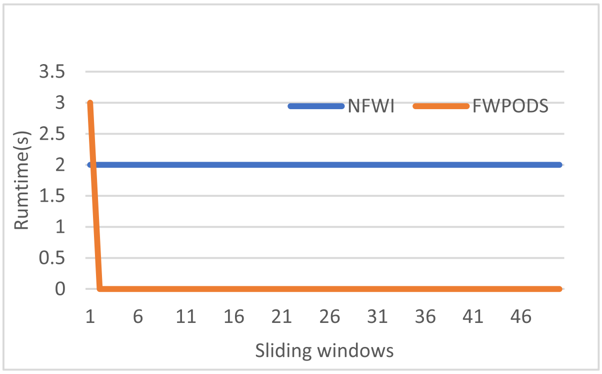


FIGURE 22. Runtime on Pumsb with ***min_ws*** = 70%, window_size = 45,000 and panel_size = 1.

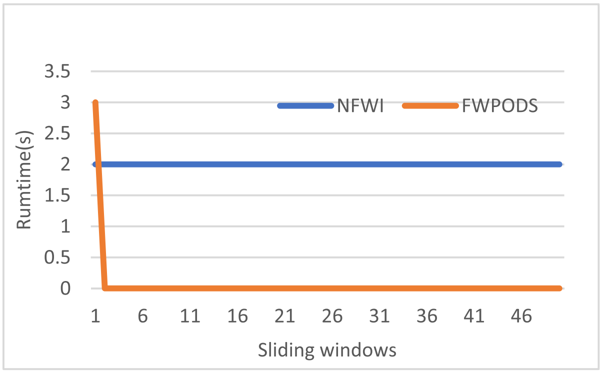


FIGURE 23. Runtime on Pumsb with ***min_ws*** = 80%, window_size = 45,000 and panel_size = 1.

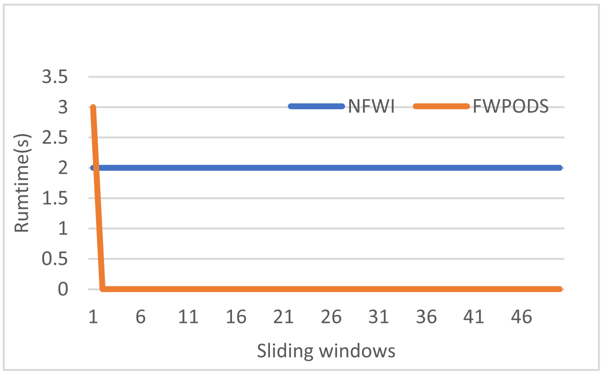


FIGURE 24. Runtime on Pumsb with ***min_ws*** = 90%, window_size = 45,000 and panel_size = 1.

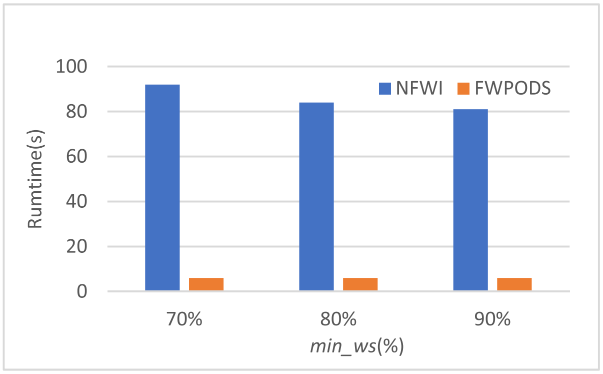


FIGURE 25. Runtimes on Pumsb in 50 sliding windows with different values of ***min_ws***.

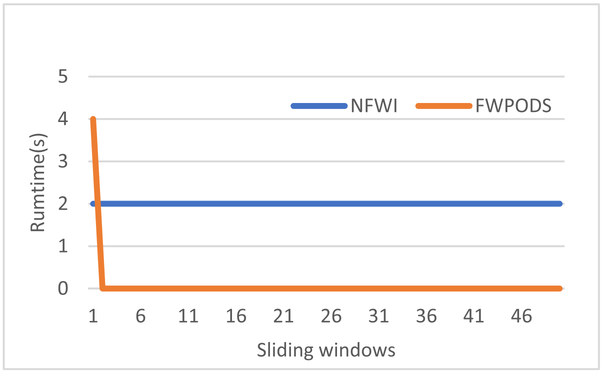


FIGURE 26. Runtime on Retail with ***min_ws*** = 0.1%, window_size = 80,000 and panel_size = 1.

B. THE PROCESSING TIME FOR TREE CONSTRUCTION MAINTENANCE

The first experiment compares the processing time of the FWPODS and NFWI [7] algorithms for tree construction and maintenance in five experimental datasets. Both algorithms are performed by three *min_ws* values for every dataset which is shown in the first three parts of each image. In our experiment, for each *min_ws* value, the window slides 50 times where the panel size is 1. The last parts of each image show the total time of 50 sliding windows with different values of *min_ws*.

The results (Figs. 10-29) are almost the same on five experimental datasets. For the first built tree, FWPODS is more time-consuming than NFWI. However, with the next sliding windows, the proposed algorithm has a barely significant runtime while that of NFWI is not improved compared with the first time. Therefore, the total time on 50 sliding-windows of FWPODS is much smaller than that of NFWI. Interestingly, the total time on 50 sliding-windows of FWPODS is

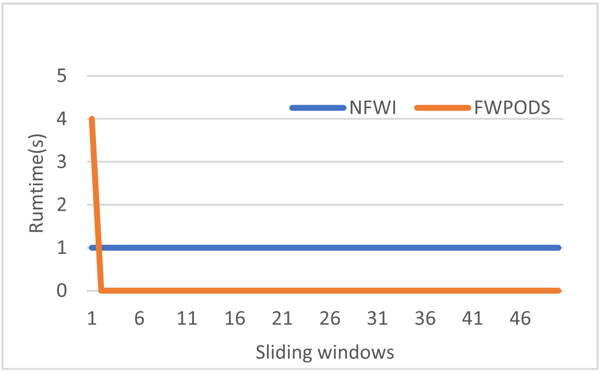


FIGURE 27. Runtime on Retail with *min_ws* = 0.2%, window_size = 80,000 and panel_size = 1.

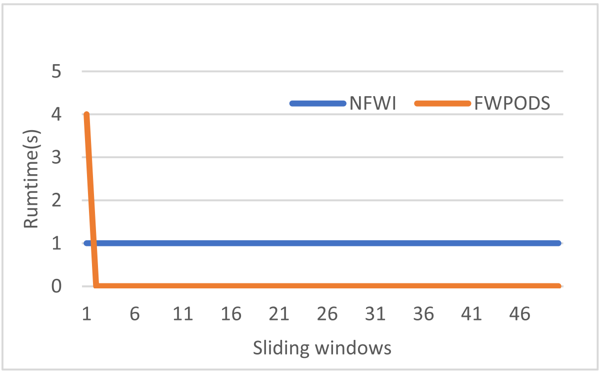


FIGURE 28. Runtime on Retail with *min_ws* = 0.3%, window_size = 80,000 and panel_size = 1.

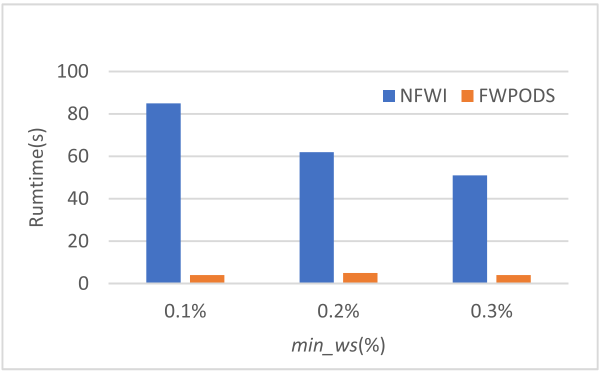


FIGURE 29. Runtimes on Retail in 50 sliding windows with different values of *min_ws*.

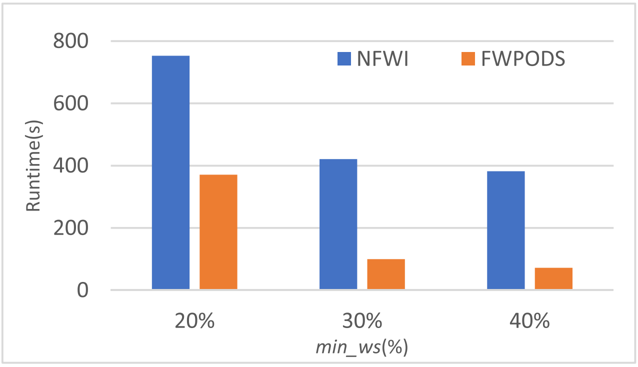


FIGURE 30. Total runtime for mining FWPs on the Accident dataset in 50 sliding windows.

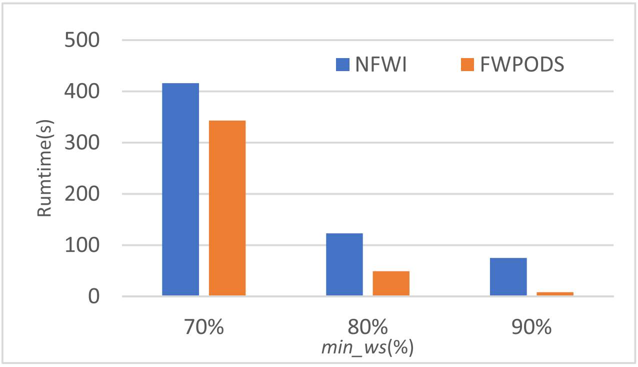


FIGURE 31. Total runtime for mining FWPs on the Connect dataset in 50 sliding windows.

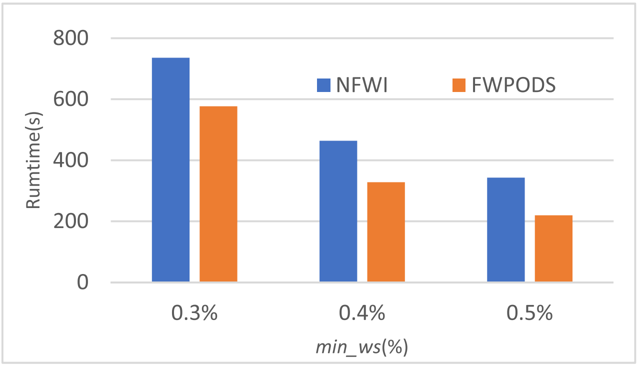


FIGURE 32. Total runtime for mining FWPs on the Kosarak dataset in 50 sliding windows.

nearly unchanged with different thresholds. Meanwhile, with a lower threshold, the total time on 50 sliding-windows of NFWI is higher.

C. RUNTIME FOR MINING FWPs OVER DATA STREAMS

This section reports the total runtime for mining FWPs over data streams on the Accident (Fig. 30), Connect (Fig. 31), Kosarak (Fig. 32), Pumsb (Fig. 33), and Retail (Fig. 34) datasets in 50 sliding windows. For very sparse datasets such as Retail (density = 0.0006) and Kosarak (density = 0.0002), FWPODS improves the runtime from 10% to 30% compared to the NFWI algorithm (Fig. 34 and Fig. 32). For the Accident, Connect, and Pumsb datasets, the proposed algorithm outperforms NFWI with all *min_ws* thresholds, and shorten the runtime from 50% to 90% compared to the NFWI algorithm (Fig. 30, Fig. 31, and Fig. 33).

D. SCALABILITY ANALYSIS

In this section, we perform scalability experiments on the Kosarak dataset with *min_ws* = 0.3% and increasing values

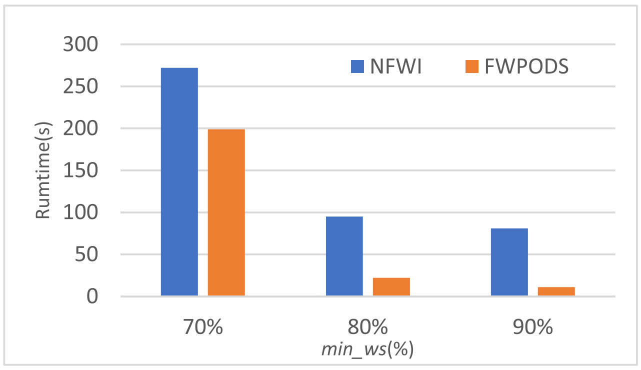


FIGURE 33. Total runtime for mining FWPs on the Pumsb dataset in 50 sliding windows.

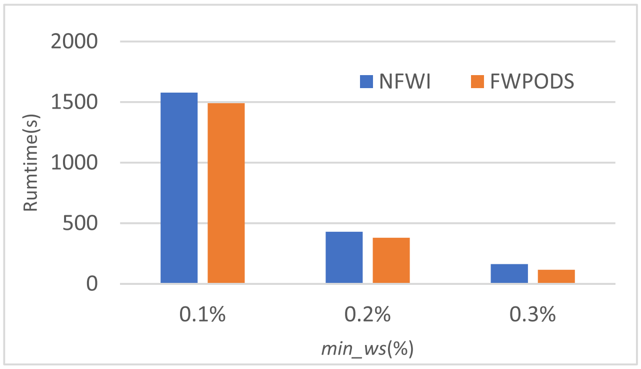


FIGURE 34. Total runtime for mining FWPs on the Retail dataset in 50 sliding windows.

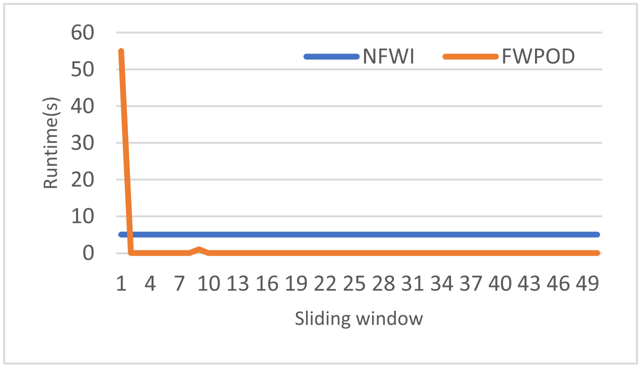


FIGURE 35. Runtime for tree construction maintenance on the Kosarak dataset with *min_ws* = 0.3%, window_size = 600,000 and panel_size = 10.

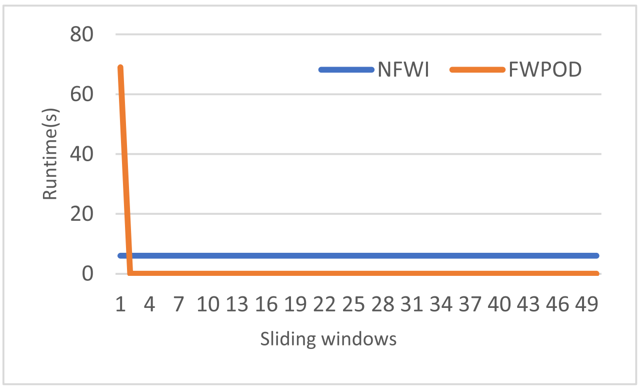


FIGURE 36. Runtime for tree construction maintenance on the Kosarak dataset with *min_ws* = 0.3%, window_size = 700,000 and panel_size = 100.

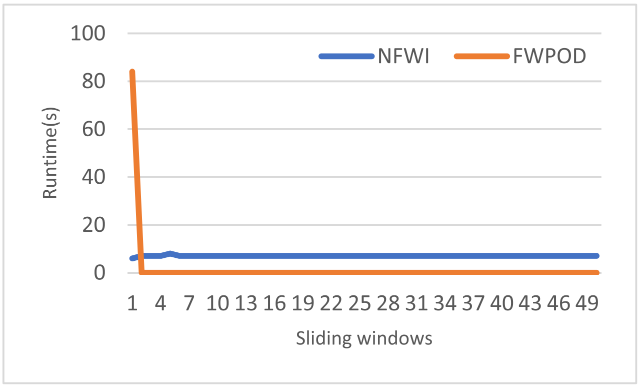


FIGURE 37. Runtime for tree construction maintenance on the Kosarak dataset with *min_ws* = 0.3%, window_size = 800,000 and panel_size = 1000.

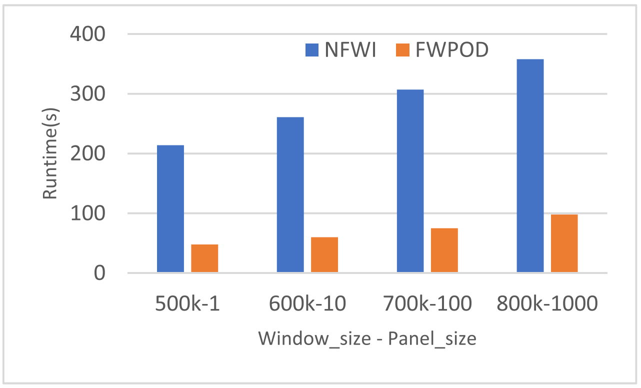


FIGURE 38. Runtimes for tree construction and maintenance on the Kosarak dataset in 50 sliding windows with *min_ws* = 0.3% and different values of window_size and panel_size.

of window and panel sizes. Specifically, the selected window sizes are 500,000 (500k), 600,000 (600k), 700,000 (700k) and 800,000 (800k), with panel sizes of 1, 10, 100 and 1000, respectively.

Observing Fig. 18 and Figs. 35-37, we see that FWPODS always uses less time than NFWI from the second sliding window and shows stability with different values of window_size and panel_size. When window_size and panel_size increase (500k to 800k and 1 to 1000) the processing time of each sliding window of FWPODS always stays the same (<1s), while that of NFWI increases steadily from 4s to 7s. Consequently, the runtimes for the tree construction maintenance and the mining process of FWPODS are much less than those of NFWI (Figs. 38-39). Therefore, FWPODS has good

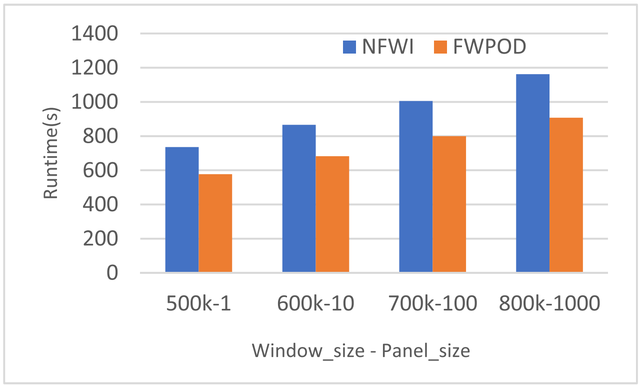


FIGURE 39. Total runtimes for mining FWPs on the Kosarak dataset in 50 sliding windows with *min_ws* = 0.3% and different values of window_size and panel_size.

scalability in terms of runtime with respect to window size and panel size.

V. CONCLUSION

This study developed an efficient approach named FWPODS for mining FWPs over data streams. Firstly, we introduced the concept of mining FWPs over a data stream via a sliding window model. Then, this study proposed an SWN-tree structure by extending the weighted node-tree (WN-tree) structure to be able to maintain the information in data streams. Next, this study developed the FWPODS algorithm for mining FWPs over data streams using the SWN-tree structure. Finally, empirical experiments were conducted to compare the performance of our approach with that of the state-of-the-art algorithm named NFWI for mining FWPs. The experimental results confirmed that FWPODS outperforms the NFWI algorithm when it runs in batch mode in a sliding window.

For future work, mining FWCPs and frequent weighted maximal patterns (FWMPs) over data streams will be studied. In addition, parallel and distributed methods for mining FWPs over data streams will be studied to take advantage of the development of hardware to improve the mining performance.

REFERENCES

[1] B. Vo, T. Le, F. Coenen, and T.-P. Hong, "Mining frequent itemsets using the N-list and subsume concepts," *Int. J. Mach. Learn. Cybern.*, vol. 7, no. 2, pp. 253–265, Apr. 2016.
[2] S. Zhang, Y. Zhang, L. Yin, T. Yuan, Z. Wu, and H. Luo, "Mining frequent items over the distributed hierarchical continuous weighted data streams in Internet of Things," *IEEE Access*, vol. 7, pp. 74890–74898, Apr. 2019.
[3] F. Tao, F. Murtagh, and M. Farid, "Weighted association rule mining using weighted support and significance framework," in *Proc. 9th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining (KDD)*, Washington, DC, USA, 2003.
[4] B. Vo, F. Coenen, and B. Le, "A new method for mining frequent weighted itemsets based on WIT-trees," *Expert Syst. Appl.*, vol. 40, no. 4, pp. 1256–1264, Mar. 2013.
[5] H. Nguyen, B. Vo, M. Nguyen, and W. Pedrycz, "An efficient algorithm for mining frequent weighted itemsets using interval word segments," *Int. J. Speech Technol.*, vol. 45, no. 4, pp. 1008–1020, Jun. 2016.
[6] G. Lee, U. Yun, and K. H. Ryu, "Mining frequent weighted itemsets without storing transaction IDs and generating candidates," *Int. J. Uncertainty, Fuzziness Knowl.-Based Syst.*, vol. 25, no. 01, pp. 111–144, Feb. 2017.
[7] H. Bui, B. Vo, H. Nguyen, T.-A. Nguyen-Hoang, and T.-P. Hong, "A weighted N-list-based method for mining frequent weighted itemsets," *Expert Syst. Appl.*, vol. 96, pp. 388–405, Apr. 2018.
[8] B. Vo, H. Bui, T. Vo, and T. Le, "Mining top-rank-*K* frequent weighted itemsets using WN-list structures and an early pruning strategy," *Knowl.-Based Syst.*, vols. 201–202, Aug. 2020, Art. no. 106064.
[9] H. Bui, B. Vo, T.-A. Nguyen-Hoang, and U. Yun, "Mining frequent weighted closed itemsets using the WN-list structure and an early pruning strategy," *Int. J. Speech Technol.*, vol. 51, no. 3, pp. 1439–1459, Sep. 2020.
[10] T. Le, B. Vo, P. Fournier-Viger, M. Y. Lee, and S. W. Baik, "SPPC: A new tree structure for mining erasable patterns in data streams," *Int. J. Speech Technol.*, vol. 49, no. 2, pp. 478–495, Feb. 2019.
[11] Y. Baek, U. Yun, H. Kim, H. Nam, G. Lee, E. Yoon, B. Vo, and J. C.-W. Lin, "Erasable pattern mining based on tree structures with damped window over data streams," *Eng. Appl. Artif. Intell.*, vol. 94, Sep. 2020, Art. no. 103735.
[12] Y. Baek, U. Yun, J. C. Lin, E. Yoon, and H. Fujita, "Efficiently mining erasable stream patterns for intelligent systems over uncertain data," *Int. J. Intell. Syst.*, vol. 35, no. 11, pp. 1699–1734, Jul. 2020.
[13] T. Le, B. Vo, and S. W. Baik, "Efficient algorithms for mining top-rank-*K* erasable patterns using pruning strategies and the subsume concept," *Eng. Appl. Artif. Intell.*, vol. 68, pp. 1–9, Feb. 2018.
[14] H. Nam, U. Yun, E. Yoon, and 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.*, vol. 529, pp. 1–27, Aug. 2020.
[15] H. Nam, U. Yun, B. Vo, T. Truong, Z.-H. Deng, and E. Yoon, "Efficient approach for damped window-based high utility pattern mining with list structure," *IEEE Access*, vol. 8, pp. 50958–50968, Mar. 2020.
[16] J.-F. Qu, P. Fournier-Viger, M. Liu, B. Hang, and F. Wang, "Mining high utility itemsets using extended chain structure and utility machine," *Knowl.-Based Syst.*, vol. 208, Nov. 2020, Art. no. 106457.
[17] J. M.-T. Wu, G. Srivastava, M. Wei, U. Yun, and J. C.-W. Lin, "Fuzzy high-utility pattern mining in parallel and distributed Hadoop framework," *Inf. Sci.*, vol. 553, pp. 31–48, Apr. 2021.
[18] U. Yun, D. Kim, E. Yoon, and H. Fujita, "Damped window based high average utility pattern mining over data streams," *Knowl.-Based Syst.*, vol. 144, pp. 188–205, Mar. 2018.
[19] J. Kim, U. Yun, E. Yoon, J. C.-W. Lin, and P. Fournier-Viger, "One scan based high average-utility pattern mining in static and dynamic databases," *Future Gener. Comput. Syst.*, vol. 111, pp. 143–158, Oct. 2020.
[20] G. S. Manku and R. Motwani, "Approximate frequency counts over data streams," in *Proc. 28th Int. Conf. Very Large Databases*, Hong Kong, Aug. 2002, pp. 346–357.
[21] J. Yu, Z. Chong, H. Lu, Z. Zhang, and A. Zhou, "A false negative approach to mining frequent itemsets from high speed transactional data streams," *Inf. Sci.*, vol. 176, no. 14, pp. 1986–2015, Jul. 2006.
[22] X. Zhi-Jun, C. Hong, and C. Li, "An efficient algorithm for frequent itemset mining on data streams," in *Proc. Adv. Data Mining Appl. Med., Web Mining, Marketing, Image Signal Mining (ICDM)*, Berlin, Germany, Jul. 2006, pp. 474–491.
[23] J. H. Chang and W. S. Lee, "Finding recently frequent itemsets adaptively over online transactional data streams," *Inf. Syst.*, vol. 31, no. 8, pp. 849–869, Dec. 2006.
[24] H. Jin Woo and W. Suk Lee, "EstMax: Tracing maximal frequent item sets instantly over online transactional data streams," *IEEE Trans. Knowl. Data Eng.*, vol. 21, no. 10, pp. 1418–1431, Oct. 2009.
[25] S.-C. Chiu, H.-F. Li, J.-L. Huang, and H.-H. You, "Incremental mining of closed inter-transaction itemsets over data stream sliding windows," *J. Inf. Sci.*, vol. 37, no. 2, pp. 208–220, Feb. 2011.
[26] M. Deypir and M. H. Sadreddini, "EclatDS: An efficient sliding window based frequent pattern mining method for data streams," *Intell. Data Anal.*, vol. 15, no. 4, pp. 571–587, Jun. 2011.
[27] M. Deypir, M. H. Sadreddini, and M. Tarahomi, "An efficient sliding window based algorithm for adaptive frequent itemset mining over data streams," *J. Inf. Sci. Eng.*, vol. 29, no. 5, pp. 1001–1020, 2013.
[28] H. Chen, L. Shu, J. Xia, and Q. Deng, "Mining frequent patterns in a varying-size sliding window of online transactional data streams," *Inf. Sci.*, vol. 215, pp. 15–36, Dec. 2012.
[29] Z. Deng, Z. Wang, and J. Jiang, "A new algorithm for fast mining frequent itemsets using N-lists," *Sci. China Inf. Sci.*, vol. 55, no. 9, pp. 2008–2030, Jul. 2012.

[30] Z. Deng and S. Lv, "PrePost^{+}: An efficient N-lists-based algorithm for mining frequent itemsets via children–parent equivalence pruning," *Expert Syst. Appl.*, vol. 42, no. 13, pp. 5424–5432, Aug. 2015.

[31] T. Le and B. Vo, "An N-list-based algorithm for mining frequent closed patterns," *Expert Syst. Appl.*, vol. 42, no. 19, pp. 6648–6657, Nov. 2015.

[32] B. Vo, S. Pham, T. Le, and Z.-H. Deng, "A novel approach for mining maximal frequent patterns," *Expert Syst. Appl.*, vol. 73, pp. 178–186, May 2017.

[33] *SPMF: A Java Open-Source Data Mining Library*. Accessed: Feb. 18, 2021. [Online]. Available: http://www.philippe-fournier-viger.com/spmf/index.php?link=datasets.php

HUONG BUI received the M.S. degree in computer science from the University of Natural Sciences, Vietnam National University—Ho Chi Minh City, in 2010. He is currently pursuing the Ph.D. degree with the Department of Computer Science, University of Information Technology, Vietnam National University—Ho Chi Minh City. His research interests include patterns mining and graph mining.

TU-ANH NGUYEN-HOANG received the M.S. and Ph.D. degrees in computer science from the University of Natural Sciences, Vietnam National University—Ho Chi Minh City, in 2002 and 2011, respectively. She is currently an Associate Professor and the Rector of the University of Information Technology, Vietnam National University—Ho Chi Minh City. Her research interests include classification and prediction, text mining, and social network mining.

BAY VO (Member, IEEE) received the B.Sc., M.Sc., and Ph.D. degrees in computer science from the University of Science, Vietnam National University, Ho Chi Minh City, Vietnam, in 2002, 2005, and 2011, respectively. He is currently an Associate Professor and the Dean of the Faculty of Information Technology, Ho Chi Minh City University of Technology, Vietnam. His research interests include association rules, classification, mining in incremental database, distributed databases, and privacy preserving in data mining.

HAM NGUYEN received the Bachelor of Science and Master of Science degrees from the University of Natural Sciences, Vietnam National University—Ho Chi Minh City, in 2001 and 2007, respectively, and the Ph.D. degree in mathematics foundation for informatics from the VNU-University of Natural Science, in 2017. He is currently a Lecturer with the Ho Chi Minh City University of Technology (HUTECH), Ho Chi Minh City, Vietnam. His research interests include data structure and algorithm, mining frequent itemsets, data mining in hierarchy database, weighted database, and quantitative database.

TUONG LE is currently a Researcher with the Informetrics Research Group, Ton Duc Thang University, Ho Chi Minh City, Vietnam. He has published more than 30 articles in prestigious journals, such as *Information Sciences*, *Expert Systems with Applications*, IEEE ACCESS, and *Engineering Applications of Artificial Intelligence*. His interests include machine learning, imbalanced learning, deep learning, business intelligence, data analysis, data mining, and pattern mining. He served as a Reviewer for several journals, like IEEE TRANSACTIONS ON CYBERNETICS, IEEE ACCESS, *Applied Soft Computing*, *Neural Computing and Applications*, *Applied Intelligence*, *PLOS ONE*, and *Engineering Applications of Artificial Intelligence*.