Efficient Discovery of Weighted Frequent Itemsets in Very Large Transactional Databases: A Re-visit

R. Uday Kiran[1,2], Amulya Kotni[3], P. Krishna Reddy[3], Masashi Toyoda[1], Subhash Bhalla[4] and Masaru Kitsuregawa[1,5]
[1]University of Tokyo, Tokyo, Japan.
[2]National Institute of Information and Communications Technology, Tokyo, Japan.
[3]International Institute of Information Technology, Hyderabad, India.
[4]University of Aizu-Wakamatsu, Aizu-Wakamatsu, Japan.
[5]Nationial Institute of Informatics, Tokyo, Japan.
{uday_rage, toyoda, masaru}@tkl.iis.u-tokyo.ac.jp, kotani@students.iiit.ac.in, pkreddy@iiit.ac.in and bhalla@u-aizu.ac.jp

***Abstract*— Weighted Frequent Itemset (WFI) mining is an important model in data mining. The popular adoption and successful industrial application of this model has been hindered by the following two obstacles: (*i*) finding WFIs is a computationally expensiveness process as these itemsets do not satisfy the downward closure property and (*ii*) lack of parallel algorithms to find WFIs in very large databases (e.g. astronomical data and twitter data). This paper makes an effort to address these two obstacles. Two pattern-growth algorithms, Sequential Weighted Frequent Pattern-growth and Parallel Weighted Frequent Pattern-growth, have been introduced to discover WFIs efficiently. Both algorithms employ three novel pruning techniques to reduce the computational cost effectively. The first pruning technique prunes some of the uninteresting items by employing a criterion known as *cutoff weight*. The second pruning technique, called *conditional pattern base elimination*, eliminates the construction of conditional pattern bases if a suffix item is an uninteresting item. The third pruning technique, called *pattern-growth termination*, defines a new terminating condition for the pattern-growth technique. Experimental results demonstrate that the proposed algorithms are memory and runtime efficient, and highly scalable as well.**

I. INTRODUCTION

Frequent itemset mining [1], [2] is an important model in data mining. Its mining algorithms discover all itemsets in the data that satisfy the user-specified minimum support (*minSup*) constraint. The *minSup* controls the minimum number of transactions that an itemset must cover within the data. Since only single *minSup* is used for the entire data, the model implicitly assumes that all items within the data have uniform frequency. However, this is the seldom case in many real-world applications. In many applications, such items appear very frequently within the data, while others rarely appear. If the frequencies of items vary a great deal, then we encounter the following two problems:

1) If *minSup* is set too high, we miss those itemsets that involve rare items in the data.
2) In order to find the itemsets that involve both frequent and rare items, we have to set *minSup* very low. However, this may cause combinatorial explosion, producing too many itemsets, because those frequent items associate with one another in all possible ways and many of them are meaningless depending upon the user and/or application requirements.

This dilemma is known as the *rare item problem*. When confronted with this problem in real-world applications, researchers have tried to find the frequent itemsets using multiple minimum supports [3], where the *minSup* of an itemset is expressed with the *minimum item support* of its items. The main limitation of this extended model is that it suffers from an open problem of determining the items' *minimum item supports*.

Cai et al. [4] introduced Weighted Frequent Itemset (WFI) model to address the rare item problem. This model finds interesting itemsets by taking into account their importance (say, *price* of an item in market-basket data) within the data. The basic model of WFIs is as follows [5]: Let $I = \{i_1, i_2, \cdots, i_n\}$, $n \geq 1$, be the set of items. Let $W = \{w_{i_1}, w_{i_2}, \cdots, w_{i_n}\}$ be the set of weights for all items in I. That is, $w_{i_k} \in W$, $w_{i_k} \in \mathbb{R}$ and $w_{i_k} \geq 0$, is the weight of $i_k \in I$, $1 \leq k \leq n$. Let W_{min} and W_{max} represent the minimum and maximum weights of all items, respectively. Thus, the **weight range** of all items in TDB is (W_{min}, W_{max}). Let $X \subseteq I$ be an **itemset** (or a pattern). If an itemset contains k items, then it is a k-itemset. A transaction $t = (tid, Y)$, where tid represents transaction-identifier and Y is an itemset. A set of transactions represents a transactional database and denoted as TDB. That is, $TDB = \{t_1, t_2, \cdots, t_m\}$, $m \geq 1$. The *support* of X, denoted as $S(X)$, represents the number of transactions containing X in TDB. The itemset X is a **frequent itemset** if $S(X) \geq minSup$, where *minSup* represents the user-specified minimum support. The weight of X, denoted as $W(X)$, represents the average weight of all items in X. That is, $W(X) = \frac{\sum_{j \in X} w_j}{|X|}$. The weighted support of X, denoted as $WS(X) = S(X) \times W(X)$. An itemset X is a **weighted frequent itemset** if $WS(X) \geq minWS$, $S(X) \geq minSup$ and $W(X) \geq minWt$, where *minWS* and *minWt* represent the user-specified *minimum weighted support* and *minimum weight* constraints, respectively. The problem of mining WFIs in

TABLE I: Transactional database

tid	itemset
1	*abcdeg*
2	*acegh*
3	*acdeg*
4	*bfi*

tid	itemset
5	*acegh*
6	*bcd*
7	*afgi*
8	*cdh*

TABLE II: Weighted frequent itemsets

Itemset	*S*	*W*
a	5	1.3
ca	4	1.2
ea	4	1.25
cea	4	1.2
ga	4	1.25
cga	4	1.2
ega	4	1.24

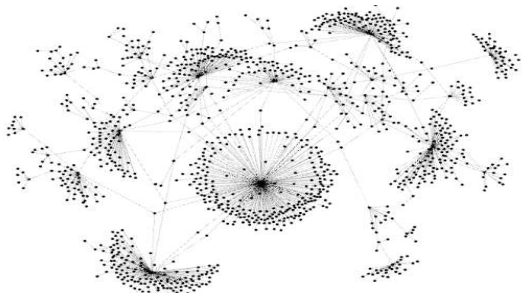
Itemset	*S*	*W*
cega	4	1.2
e	4	1.2
d	4	1.3
cd	4	1.2
g	5	1.2
eg	4	1.2

TDB is to discover all WFIs that satisfy the *minWt*, *minSup* and *minWS* constraints. (The association rules generated from the WFIs are known as weighted association rules. For brevity, we are not discussing about these rules in this paper. One can reduce the number of input parameters by considering $minSup = minWS$.)

Example 1: Table I shows a transactional database with the set of items $I = \{abcdefghi\}$. Let the weights of the items a, b, c, d, e, f, g, h and i be 1.3, 1.2, 1.1, 1.3, 1.2, 1.5, 1.2, 1.1 and 1.3, respectively. The W_{min} and W_{max} are 1.1 and 1.5, respectively. Therefore, the weight range is $(1.1, 1.5)$. The set of items 'a' and 'c', i.e., 'ac' is an itemset. This itemset contains 2 items. Therefore, it is a 2-itemset. The itemset 'ac' appears in 4 transactions. Therefore, the *support* of ac, i.e., $S(ac) = 4$. If the user-specified $minSup = 4$, then ac is a frequent itemset. The weight of this itemset, i.e., $W(ac) = \frac{w_a + w_c}{|ac|} = \frac{1.3+1.1}{2} = 1.2$. The weighted support of ac, i.e., $WS(ac) = 4 \times 1.2 = 4.8$. If the user-specified $minWS = 4$ and $minWt = 1.2$, then the frequent itemset ac is also a weighted frequent itemset because $WS(ac) \geq minWS$ and $W(ac) \geq minWt$. The set of all WFIs discovered from Table I are shown in Table II.

The weights for items can be either specified by the user or determined using information theory measures (e.g. entropy [6] and Term Frequency-Inverse Document Frequency). Some of the practical applications of WFI model are market-basket analytics [4] and spectral signature analytics in astronomical databases [6]. Another important application of this model is event analytics in social datasets (e.g. Twitter). Events (or topics) in general are multifaceted covering various topics. Some topics can be frequent, while others can be relatively infrequent (or rare) in the data. The WFI mining on social datasets can facilitate us to find the correlations between the frequent and rare topics. This information can further be used for various purposes, such as understanding the information cascades [7] and topic summarization [8]. For instance, WFI mining on a Twitter corpus related to Manchester Arena bombing has revealed that the words "explosion" and "injured" have occurred together in 1,852 tweets out of 97,000 tweets. Further analytics on these tweets, such as topology (see Fig. 1a) and popular tweets (see Fig. 1b), can be found useful to the analysts. Topology is crucial to understand the information cascade.

Cai et al. [4] discussed Apriori algorithms to find WFIs. Unfortunately, these algorithms suffer from the performance issues involving the generation of huge number of candidate itemsets and multiple scans on the database. Yun and John



(a)

S.no.	Tweets
1	Police in Manchester England say atleast 19 dead & 50 jured amid reports of explosion at Ariana Grande's concert #ManchesterBombing
2	My prayers to those injured after explosion at the Ariana Grande concert in the UK #PrayforManchester
3	Ariana Grande says she is broken and suspends #DWTour after explosion in Manchester that left 22 dead and 59 injured

(b)

Fig. 1: Analytics on the tweets containing the words 'explosion' and 'injured'. (a) tweet-retweet network of users and (b) top-3 retweets containing these two words

[5] discussed a pattern-growth algorithm, called Weighted Frequent Itemset Mining (WFIM), to find WFIs. Since the WFIs do not satisfy the downward closure property, the WFIM finds candidate weighted frequent 1-itemsets (CWFIs), constructs WFIM-tree constituting of CWFIs, and recursively mines the entire tree by building the conditional pattern bases (CPBs) for every item in the tree. In other words, WFIM basically extends Frequent Pattern-growth (FP-growth [9]) to find WFIs. The criterion used to find CWFIs is known as "*weight upper bound*," and it involves pruning all uninteresting itemsets whose product of *support* and maximum weight of all items in the database is less than the user-specified *minWS*.

Since the WFIs discovered by the basic model do not satisfy the downward closure property, researchers have discussed several alternative weighing functions, such as maximum weight and transaction weight, to find WFIs. Each weighing function has a selection bias that justifies the significance of one itemset over another. As a result, there exists no best model to discover WFIs in any given database. In this paper, we focus on efficient discovery of WFIs using the basic model as this model is widely used in practical applications.

Pei et al. [10] introduced convertible constraints, and showed that weighted average satisfies the convertible anti-monotonic property. The authors have also showed that measures satisfying the convertible anti-monotonic property can be directly pushed into Frequent Pattern-growth (FP-growth) algorithm [9]. It has to be noted that although weighted average satisfies the convertible anti-monotonic property, we cannot simply use FP-growth algorithm to discover WFIs. It is because the WFI model has to take into account weighted support, which is a product of *support* and *weighted average* of all items within an itemset.

Example 2: In Table II, it can be observed that the item *c* is not a weighted frequent item. However, it can be observed that this item has still generated weighted frequent itemsets by combining with other items. Thus, the existing downward closure property based FP-growth algorithm cannot be directly used for finding the WFIs although weighted average satisfies the convertible anti-monotonic property.

This paper argues that finding WFIs using WFIM is costly because of two main reasons: (*i*) many uninteresting items whose supersets cannot be WFIs can still satisfy the "*weight upper bound criterion*" and (*ii*) recursive mining of entire WFIM-tree increases the computational cost of WFIM because the algorithm has to search the supersets of those uninteresting items (or itemsets) that cannot be WFIs. It is thus important to generate WFIs without taking into account many uninteresting itemsets.

In the era of BigData, parallel algorithms are playing a key role in finding useful information in very large databases. In the field of pattern mining, researchers have discussed many parallel frequent pattern mining algorithms using Map-Reduce framework [11], [12]. These parallel frequent pattern mining algorithms can be extended to mine WFIs. However, as we discussed in the above paragraphs, such extended algorithms are costly as they generate WFIs by taking into account many uninteresting itemsets.

With this motivation, this paper revisits the problem of finding the WFIs in a transactional database. In this paper, we show that WFIs satisfy the *sorted closure property* [3]. Two pattern-growth algorithms, called 'Sequential Weighted Frequent Pattern Growth' (SWFP-growth) and 'Parallel Weighted Frequent Pattern Growth' (PWFP-growth), have been introduced to find WFIs. Both algorithms employ three novel pruning techniques to reduce the computational cost effectively. The first pruning technique, called "cutoff weight," eliminates uninteresting items (or itemsets) whose supersets cannot be WFIs. The second pruning technique, called "conditional pattern base elimination," prevents the construction of conditional pattern bases for the uninteresting (suffix) items. The third pruning technique, called "pattern-growth termination," defines the terminating condition for the pattern-growth technique. This pruning technique eliminates the recursive mining of entire *tree* (as by done by existing FP-growth-like algorithms). Thus, reducing the computational cost effectively. Experimental results demonstrate that the proposed algorithms are efficient.

The rest of the paper is organized as follows. Section 2 describes the related work. Section 3 describes the WFIM algorithm. Section 4 describes the performance issues of WFIM and introduces the proposed SWFP-growth algorithm. Section 5 describes the proposed PWFP-growth algorithm. Section 6 reports results. Finally, Section 7 concludes the paper with future research directions.

II. Related Work

In this section, we first discuss the related work on finding WFIs in the data. Next, we describe the works on various properties used to reduce the search space in itemset mining.

A. Weighted frequent itemset mining

Cai et al. [4] introduced weighted frequent itemset mining as an intermediary step to find weighted association rules in a transactional database. Two Apriori algorithms, called MinWAL(O) and MinWAL(M), have been discussed for finding WFIs. Unfortunately, these two algorithms suffer from the performance issues involving multiple database scans and generation of too many candidate itemsets. Yun and John [5] discussed a pattern-growth algorithm, called WFIM, to find the weighted frequent itemsets. Cai et al. [6] used a variant of WFIM algorithm to find weighted frequent itemsets in astronomical databases. This algorithm uses an entropy based weighting function to determine the interestingness of an itemset. The rules generated from these itemsets are called as stellar spectra association rules. Although WFIM and its variants do not suffer from the performance issues as that of the Apriori algorithms, they are still inadequate to find the WFIs effectively. It is because such algorithms have to take into account many uninteresting itemsets whose supersets may not generate any weighted frequent itemset. In this paper, we propose an improved pattern-growth algorithm that efficiently prunes uninteresting itemsets whose supersets may not generate any weighted frequent itemset.

In the literature, researchers have extended weighted frequent itemset mining by taking into account other parameters. Tao et al. [13] proposed a weighted association rule model by taking into account the weight of a transaction. An Apriori-like algorithm, called WARM (Weighted Association Rule Mining) algorithm, was discussed to find to the itemsets. Vo et al. [14] proposed a Weighted Itemset Tidset tree (WIT-tree) for mining the itemsets and used a Diffset strategy to speed up the computation for finding the itemsets. Lin et al. [15] studied the problem of finding weighted frequent itemsets by taking into account the occurrence time of the transactions. The discovered itemsets are known as recency weighted frequent itemsets. Furthermore, Lin et al. [16] extended the basic weighted frequent itemset model [4] to handle uncertain databases. Chowdhury et al. [17] discussed a weighted frequent itemset model with an assumption that weights of items can vary with time and proposed the algorithm AWFPM (Adaptive Weighted Frequent Pattern Mining). The pruning techniques proposed in this paper can be extended to some of these extended weighted frequent itemset models. However, in this paper, we confine to the basic weighted frequent itemset model due to page limitation.

Utility itemset mining is another important model in data mining [18]. This model takes into account internal utility (say, number of times an item has appeared within a transaction) and external utility (say, price of an item) of items within the database and tries to find itemsets that have highly utility value. Since some of the utility functions as discussed in [10] satisfy the sorted closure property, the proposed pruning techniques can be extended to find utility itemsets effectively. As a part of future work, we would like to extend the proposed pruning techniques to utility itemset mining.

B. Various properties used to reduce the search space in itemset mining

Reducing the search space is an important problem in itemset mining. Aggrawal et al. [1] discussed **downward closure property** to reduce the search space in frequent itemset mining. This property was widely used by many itemset mining algorithms. Liu et al. [3] discussed **sorted closure property** to reduce the search space for finding the frequent itemsets with multiple minimum supports. Pei et al. [10] discussed convertible anti-monotonic property, convertible monotonic property and succinct properties to discover constraint-based itemsets. In this paper, we show that weighted frequent itemsets satisfy the sorted closure property, and use this property to reduce the search space effectively.

Over the past two decades, several improvements have been suggested to improve the performance of FP-growth algorithm. Unfortunately, most of these improvements cannot be employed in the proposed algorithm because they were based on the downward closure property.

III. WFIM ALGORITHM

The working of WFIM involves the following two steps: (*i*) compress the database into WFIM-tree and (*ii*) recursively mine the entire tree to find all WFIs. The structure of WFIM-tree is same as that of the FP-tree and mining procedure is same as that of the FP-growth. However, since WFIs do not satisfy the downward closure property, WFIM finds candidate weighted frequent itemsets (CWFIs) and generate WFIs from CWFIs. The CWFIs are generated by employing the following pruning conditions: (*i*) $W(X) < minWt$ and $S(X) < minSup$ and (*ii*) $S(X) \times W_{max} < minWS$. Since conditional pattern base represents the sub-database containing the items in suffix itemset, W_{max} is replaced with the lowest *weight* of all items in the suffix itemset. We briefly describe the working of WFIM using Table I. Let $minSup = minWS = 4$ and $minWt = 1.2$.

Before scanning the database, all items are inserted into the WFIM-list with *supports* set to 0 (see Fig. 2(a)). Next, the *supports* of all items in the WFIM-list is determined by scanning the entire database (see Fig. 2(b)). Next, candidate weighted frequent items are generated using the above mentioned pruning conditions, and sorted in ascending order of their weights (see Fig 2(c)). In the next step, the prefix-tree in WFIM-tree is constructed by performing another scan on the database (see Figure 2(d)). The construction procedure is same as that of the FP-tree [9]. Finally, the **entire** WFIM-tree is recursively mined to find CWFIs. The WFIs are generated from CWFIs. The mining procedure is same as that of the FP-growth, and is summarized in Fig 3.

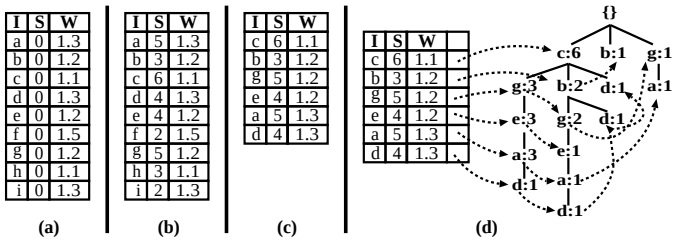


Fig. 2: Construction of WFIM-tree. (a) WFIM-list before scanning the database (b) WFIM-list after scanning the entire database (c) Final WFIM-list (d) Final WFIM-tree

IV. SEQUENTIAL WEIGHTED FREQUENT PATTERN-GROWTH

In this section, we first report the performance issues of WFIM. Next, we describe the basic idea for generating the WFIs effectively. Finally, we introduce the SWFP-growth.

A. Performances issues of WFIM

The WIFM algorithm generates WFIs by taking into account many uninteresting itemsets whose supersets cannot be WFIs. The reasons are as follows:

1) Many uninteresting items can satisfy the weight upper bound criterion. For instance, WFIM considers *b* in Table I as a CWFI because $S(b) \times W_{max} \geq minWS$, However, neither *b* nor its supersets are WFIs as shown in Table II.
2) The WFIM constructs CPBs for the uninteresting items in the WFIM-list, and mines all those CPBs recursively. Unfortunately, all itemsets generated during this recursive mining process are useless because none of them will be WFIs.
   Example 3: The WFIM constructs CPBs for *b* (see Fig. 3), and mines it recursively to find WFIs. All the itemsets generated during this recursive mining process are useless because none of them will be WFIs.
3) The WFIM terminates only when the WFIM-list is empty. This terminating condition will increase the computational cost because CPBs of the suffix items with weights less than $minWt$ will not generate WFIs.

In the next subsection, we describe the basic idea to reduce the computational cost of finding the WFIs.

B. Basic idea

We observed that WFIs discovered using weighted average satisfy the *sorted closure property* [3]. The correctness is based on Property 1 and shown in Lemmas 1 and 2. We now discuss three pruning techniques based on this property.

I	S	W	Cond. Pattern Base	Cond. WFIM-Tree	Weighted Frequent Patterns
c	6	1.1	{}	–	–
b	3	1.2	{c : 2}	–	–
g	5	1.2	{c : 3}, {cb : 1}	<c : 4>	–
e	4	1.2	{cg : 3}, {cbg : 1}	<c g : 4 >	{eg : 4, 1.2}
a	5	1.3	{cge : 3}, {cbge : 1}, {g : 1}	<c g e : 4>, <g : 1>	{ae : 4, 1.25}, {aeg : 4, 1.24}, ... ,{ag : 5, 1.25}, ...
d	4	1.3	{cgea : 1}, {cbgea : 1}, {cb : 1}, {c : 1}	<c : 4>	{dc : 4, 1.2}

Fig. 3: Mining Weighted Frequent Itemsets using WFIM Algorithm

Property 1: Let $Y = \{i_1, i_2, \cdots, i_k\}$, $1 \leq k \leq n$, be a sorted itemset with $W(i_1) \geq W(i_2) \geq \cdots \geq W(i_k)$. If $X = Y \cup i_{k+1}$, $W(i_k) \geq W(i_{k+1})$, then $sup(Y) \geq sup(X)$ and $W(Y) \geq W(X)$.

Lemma 1: Let $Y = \{i_1, i_2, \cdots, i_k\}$, $1 \leq k \leq n$, be a sorted itemset with $W(i_1) \geq W(i_2) \geq \cdots \geq W(i_k)$. If $X = Y \cup i_{k+1}$, $W(i_k) \geq W(i_{k+1})$, then $sup(Y) \times W(Y) \geq sup(X) \times W(X)$.

Proof: Based on Propery 1, it turns out that if $Y \subset X$ and Y contains i_1, then $sup(Y) \geq sup(X)$ and $W(Y) \geq W(X)$. Thus, $sup(Y) \times W(Y) \geq sup(X) \times W(X)$. Hence proved. ■

Lemma 2: **(Sorted closure property of X.)** Let $X = \{i_1, i_2, \cdots, i_k\}$, $1 \leq k \leq n$, be a sorted itemset with $W(i_1) \geq W(i_2) \geq \cdots \geq W(i_k)$. If X is a weighted frequent pattern, then $\forall Y \subset X$ and $i_1 \in Y$, Y is a weighted frequent pattern.

Proof: The correctness is straight forward to prove from Lemma 1. ■

1) Pruning Technique 1: (cutoff-weight criterion).: If X is a WFI, then the item with maximum weight in X is also a weighted frequent item (i.e., sorted closure property). Henceforth, the maximum weight that can be achieved by any weighted frequent itemset will always be less than or equivalent to the maximum weight of all weighted frequent items (or 1-itemsets). We call the maximum weight of all weighted frequent items as *Cutoff weight*, and is defined as follows.

Definition 1: Let $I_1 \subseteq I$ be the set of all weighted frequent items in TDB. The cutoff weight, $CW = max(i_j | \forall i_j \in I_1)$.

Example 4: The set of all weighted frequent items in Table I are c, g, e, a and d. The cutoff weight CW = $max(W(c), W(g), W(e), W(a), W(d))$ = 1.3.

We now introduce the following criterion to prune uninteresting itemsets: "*Prune an itemset X if $S(X) \times CW < minWS$, because neither X nor its supersets can be WFIs.*"

Example 5: The weight upper bound criterion of WFIM considers b as a CWFI. However, the proposed criterion considers b as an uninteresting item as $S(b) \times CW < minWS$. Thus, the proposed criterion is tighter than the weight upper bound criterion.

2) Pruning technique 2: conditional pattern base elimination.: Since weighted frequent itemsets satisfy the sorted closure property, we will be constructing *tree* by taking into account some uninteresting items whose product of *support* and *CW* has satisfied the *minWS*. This pruning technique says that if the suffix itemset is an uninteresting itemset, then prevent the construction of its *CPB* as no further WFIs can be generated. The correctness of our argument is based on Lemma 3 and shown in Lemma 4.

Lemma 3: Let T be a tree constructed in ascending order of items' weights. If a be the suffix item and b is an item present in its conditional pattern base, then $S(a) \times W(b) \geq S(ab) \times W(ab)$.

Proof: Since tree is constructed in ascending order of items' weights, $W(a) \geq W(b)$. Thus, the $W(ab) \leq W(a)$. Similarly, $S(ab) \leq S(a)$. Thus, $S(a) \times W(b) \geq S(ab) \times W(ab)$. Hence proved. ■

Lemma 4: Let T be a tree constructed in ascending order of items' weights. Let a be the suffix item and b be another item that is present in the conditional pattern base of a. If a is an uninteresting item (i.e., $S(a) \times W(a) < minWS$), then ab is also an uninteresting itemset.

Proof: According to Lemma 3, $S(a) \times W(b) \geq S(ab) \times W(ab)$. Thus, $S(ab) \times W(ab) \leq S(a) \times W(b) < minWS$. In other words, ab is an uninteresting itemset. Hence proved. ■

3) Pruning technique 3: pattern-growth termination.: This pruning technique states that if the weight of a suffix item in the *tree* is less than $minWt$, then terminate the pattern-growth technique as no further WFIs can be generated.

Since the *tree* is constructed in ascending order of items' weights, it turns out that the weights of all items in the conditional pattern base of a suffix item will always be less than or equal to the weight of suffix item. Thus, if the weight of suffix item is less than $winWt$, then the weight of all items in its conditional pattern base will always be less than or equal to $winWt$. In other words, if the suffix item fails to satisfy the $minWt$, then all itemsets that can be generated from its conditional pattern base will also fail to satisfy the $winWt$. Thus, there is no need for constructing the conditional pattern bases for the suffix items that have weight less than $minWt$. Moreover, since the *tree* is constructed in weight ascending order, we can immediately stop the pattern-growth technique once we encounter a suffix item with weight less than $minWt$. It is because the next suffix item(s) will also have weight less than $minWt$. We call this pruning technique as pattern-growth termination.

Example 6: Let x, y and z be three sorted items in a list with *weights* 1.1, 1.2, and 1.3, respectively. Let us consider z as a suffix item. The conditional pattern base of z can constitute of items, x and y. According to Property 1, it turns out that $W(z) \geq W(zy) \geq W(zyx)$ and $W(z) \geq W(zx) \geq W(zyx)$. If $minWt = 1.5$, then z is not a WFI because $W(z) < minWt$. Similarly, all itemsets generated from the

conditional pattern base of z will also have weight less than $minWt$. Thus, they are also not weighted frequent itemsets. The weight of y, i.e., $W(y) < W(z) < minWt$. Thus, y is not a weight frequent itemset. Similarly, all itemsets generated from the conditional pattern base of y will also have weight less than $minWt$. Thus, they are also not weighted frequent itemsets. The same can be said about the item x. So forth, when we find that item z has weight less than $minWt$, we can stop the pattern-growth technique as all other items in the list will not generate any weighted frequent itemset.

Algorithm 1 SWFP-list(TDB, $minWS$, $minWt$, $weights$ of items')

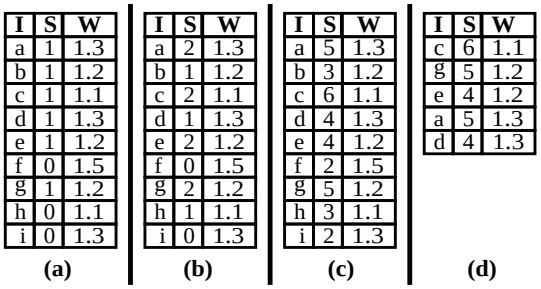
1: Insert all items into SWFP-list with their weights and support set to 0
2: **for** each transaction $t \in TDB$ **do**
3: **for** each item i in transaction t **do**
4: increase the support of i in SWFP-list by 1
5: Sort the SWFP-list in ascending order of items' weights. Find the weighted frequent items, and choose the maximum weight among all weighted frequent items as cutoff-weight (CW).
6: **for** each item a_i in SWFP-list **do**
7: **if** (($sup(a_i) < minSup$) and ($weight(a_i) < minWt$)) or ($sup(a_i) \times CW < minWS$) **then**
8: Prune a_i from the SWFP-list
9: Consider the remaining items in the SWFP-list as the candidate weighted frequent items, and construct SWFP-tree. The construction procedure for SWFP-tree is same as that of the FP-tree [9] with the key difference that *support* information is stored only at the tail-node of a sorted transaction.

C. SWFP-growth

The working of SWFP-growth involves the following two steps: (*i*) compress the database into SWFP-tree and (*ii*) recursively mine the tree to discover WFIs. We now explain the structure, construction and mining of SWFP-tree.

1) Structure of SWFP-tree.: The structure of SWFP-tree is similar to that of FP-tree. However, to achieve memory efficiency, the proposed tree records support information only at the tail node of a sorted transaction. Moreover, items in SWFP-tree are arranged in weight ascending order of items.

2) Construction of SWFP-tree.: Since WFIs do not satisfy the downward closure property, candidate weighted frequent items play a key role in finding WFIs effectively. The procedure for finding candidate weighted frequent items is shown in Algorithm 1. Briefly, this algorithm inserts all items in to the SWFP-list with their weights and *support* set to 0 (see Fig. 4 (a)). Next, the *support* of all items in the list is updated by scanning every transaction in the database (see Fig. 4 (b)). Next, uninteresting items whose supersets cannot be WFIs are removed from the list by adopting the following criterions: (*i*) prune an item i if $S(i) < minSup$ and $w(i) < minWt$ and (*ii*) prune an item i if $S(i) \times CW < minWS$.



I	S	W
a	1	1.3
b	1	1.2
c	1	1.1
d	1	1.3
e	1	1.2
f	0	1.5
g	1	1.2
h	0	1.1
i	0	1.3

(a)

I	S	W
a	2	1.3
b	1	1.2
c	2	1.1
d	1	1.3
e	2	1.2
f	0	1.5
g	2	1.2
h	1	1.1
i	0	1.3

(b)

I	S	W
a	5	1.3
b	3	1.2
c	6	1.1
d	4	1.3
e	4	1.2
f	2	1.5
g	5	1.2
h	3	1.1
i	2	1.3

(c)

I	S	W
c	6	1.1
g	5	1.2
e	4	1.2
a	5	1.3
d	4	1.3

(d)

Fig. 4: Construction of SWFP-list. (a) After scanning first transaction (b) After scanning second transaction (c) After scanning entire database (d) Final list of candidate weighted items

The remaining items in the list are considered as candidate weighted frequent items and sorted in ascending order of their weights (see Fig. 4 (c)).

After finding candidate weighted frequent items, we conduct another scan on the database and construct the prefix-tree of the SWFP-tree. Fig. 5(a) shows the prefix-tree generated after scanning the first transaction. It can be observed that only the tail-node a carries the support count. Fig. 5(b) shows the prefix-tree generated after scanning the second transaction. Fig. 5(c) shows the SWFP-tree generated after scanning the entire database.

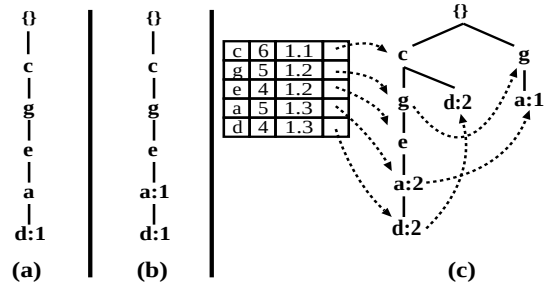


Fig. 5: Construction of SWFP-tree. (a) After scanning first transaction (b) After scanning second transaction (c) Final tree after scanning the entire database

Algorithm 2 SWFP-growth(SWFP-tree, α)

1: Select the last item in the SWFP-list
2: **for** each item a_i in header of SWFP-Tree **do**
3: **if** $W(a_i) < minWt$ **then**
4: exit();
5: **else**
6: Generate pattern $\beta = a_i \cup \alpha$;
7: **if** β is a WFI **then**
8: Construct β's conditional SWFP-tree, $Tree_\beta$;
9: **if** $Tree_\beta \neq 0$ **then**
10: Call SWFP-growth($Tree_\beta$, β);
11: Remove a_i from the tree and add a_i's support to its parent node's support

3) Mining Weighted Frequent Patterns.: The SWFP-tree is mined as follows. Start from length-1 itemsets (as an initial suffix item). If this item is a weighted frequent itemset, construct its conditional pattern base (a sub-database, which consists of the set of prefix paths in the tree with the suffix item), then construct its construct its conditional SWFP-tree and mine it recursively. Pattern growth is achieved by concatenating the suffix item with the weighted frequent itemset generated from the conditional SWFP-tree. Next, the initial suffix item is pruned from the original SWFP-tree by pushing its support to the corresponding parent nodes. The above process is repeated until SWFP-list is empty or the weight of the suffix item is less than $minWt$. Please note that if a suffix itemset is an uninteresting itemset, then we will not construct its conditional pattern base. We simply prune the itemset from the original tree by pushing its support counts to the corresponding parent nodes. Fig. 6 summarizes the mining of SWFP-tree shown in Fig. 5 (c).

After finding the weighted frequent itemsets, weighted association rules are generated by directly applying the association rule mining procedure [1] on weight frequent itemsets. In this paper, we are not discussing this procedure for brevity.

V. Parallel Weighted Frequent Pattern-Growth

Due to page limitation, we briefly explain the PWFP-growth algorithm. The working of PWFP-growth is similar to that of the Parallel Frequent Pattern-growth (PFP-growth)[11]. However, the key differences are as follows: (*i*) The PWFP-growth algorithm finds WFIs by taking into account all items that satisfy the cutoff weight criterion. (*ii*) The *trees* in worker machines are constructed in weight descending order and (*ii*) During the recursive mining of a *tree* in each worker machine, *conditional pattern base elimination* and *pattern-growth terminating condition* are employed to reduce the computational cost effectively.

The extraction of weighted frequent itemsets in parallel requires two database scans. Each scan consists of one Map-Reduce job. The transactional database is divided into multiple shards (partitions) and each partition is allotted to a machine so that processing can be done in parallel. We have employed PFP-growth procedures for data partition and assigning to worker machines. The proposed algorithm consists of 2 steps:

1) Finding candidate items: Each worker machine scans the transactions and outputs key-value pairs with key as the item and value as 1 ($< item, 1 >$). In the reduce phase, these key-values are aggregated by the master machine to derive *support* count of the items. Next, the master machine sorts the items in weight ascending order and finds candidate items by employing the cutoff-weight criterion. The procedure for finding candidate items is shown in lines 5 to 8 in Algorithm 1. Next, the master machine assigns ranks to the candidate items such that item with the least weight gets the lowest rank. Let the list of these candidate items be called WF-list. The final WF-list is transferred for all worker machines.

2) Construction and Mining of WF-trees: In the second database scan, for each transaction the items which are not present in the WF-list are filtered, translated into their ranks and sorted in ascending order. The sub-patterns (conditional transactions) for each transaction are extracted and assigned to a machine based on the hash function: (rank[item]% Num of machines). Here, item is the last item in a sub-pattern. **Subpatterns are constructed only if the last item is a weighted frequent item (i.e., conditional pattern base termination).** Once we encounter an item which is not a weighted frequent item, we stop extracting sub-patterns from that transaction as the items after it will have weight less than *minW* (i.e., pattern-growth termination). The assigned machine is stored in a hash-table for future look-up. The hash function gives a machine-id for which the pattern is responsible for further computation. Thus, the sub-patterns for each weighted frequent item in every transaction are generated and sent to the corresponding machine. Each sub-pattern is emitted as a key-value pair, with key as the machine-id and value as sub-pattern. Now, the reduce function is implemented with machine-id as key, hence all the conditional transactions with same machine-id are processed at one machine. Independent local WF-trees are constructed by inserting all the sub-patterns into the tree in the same order as WF-list. The process of tree construction is same as SWFP-tree construction. Since the trees are constructed from the sub-patterns itself, during conditional pattern building, communication is not required between the machines. The workers already have the information required to build the conditional pattern trees. This way the mining process can be done in parallel without any communication cost.

Parallel mining of weighted frequent itemsets is similar to the mining process of FP-Growth algorithm but each worker machine performs the mining process only for those suffix items for which it is responsible for computation. The prefix tree is constructed for a chosen suffix item by inserting the prefix sub paths of the nodes of the selected item. The conditional tree is constructed from the prefix tree by removing the nodes which satisfy the pruning conditions. This process is repeated for all the items assigned to each worker node. Finally, the itemsets extracted by all the worker nodes are gathered at the master node. Here, it should be noted that the process of tree construction and mining is done only for weighted frequent items and the items having weight less than *minW* are neglected as they cannot generate weighted frequent itemsets as suffix items.

VI. Experimental Results

In this section, we evaluate the performance of SWFP-growth and PWFP-growth algorithms on various databases, and show that the proposed algorithms are efficient. The algorithms WFIM and SWFP-growth were written in Python and run on a 2.66 GHz machine having 16 GB of memory. The operating system used in this machine is Ubuntu 14.04.

Since there exists no parallel algorithm to find WFIs, we have extended the existing Parallel Frequent Pattern-growth (PFP-growth) algorithm [11] to find WFIs using "weight

I	S	W	Cond. Pattern Base	Cond. WFIM-Tree	Weighted Frequent Patterns
g	5	1.2	{c : 4}	–	–
e	4	1.2	{cg : 4	<c g : 4 >	{eg : 4, 1.2}
a	5	1.3	{cge : 4}, {g : 1}	<c g e : 4>, <g : 1>	{ae : 4, 1.25}, {aeg : 4, 1.24}, ... ,{ag : 5, 1.25}, ...
d	4	1.3	{cgea : 2}, {c : 2}	<c : 4>	{dc : 4, 1.2}

Fig. 6: Mining Weighted Frequent Itemsets using SWFP-growth

upper bound criterion." We call this extended algorithm as Parallel Weighted Frequent Itemset Mining (PWFIM). We will use this naïve algorithm to evaluate the PWFP-growth algorithm. Both algorithms are written in Python using Apache Spark architecture and the experiments are conducted on Amazon Elastic Map-Reduce cluster, with each machine having 8GB memory. The runtime is measured in seconds and specifies the total execution time of the spark job. The data shuffled is measured in KB and measures the communication among the machines.

The experiments have been conducted using both synthetic (**T10I4D100K**) and real-world (**Connect**, **Mushroom**, and **Twitter**) databases. The synthetic database was generated by using the IBM data generator [1]. This data generator is widely used for evaluating association rule mining algorithms. The **T10I4D100K** database contains 870 items with 100,000 transactions. The **Connect** and **Mushroom** databases have been downloaded from the Frequent Itemset MIning (FIMI) repository. The **Connect** database contains 129 items with 67557 transactions. The **Mushroom** database contains 119 items with 8124 transactions. The **Twitter** database constitutes of 97,000 tweets collected from 22-may-2017 to 23-may-2017. These tweets are related to 2017 Manchester Arena bombing. We have created a database by considering top 1000 frequent English words. Please note that sequential algorithms were evaluated using all of the above mentioned databases. On the other hand, sensitive Twitter data is not used for evaluating the parallel algorithms as Amazon EC2 is a public cloud.

We have used random number generator to assign weights to items. Similar procedure is used for evaluating the WFI mining algorithms. Please note that in our case study, we have employed TF-IDF to assign weights to items in Twitter database.

Fig. 7 (a)-(d) show the number of WFIs generated in various databases at different *minWS* and weight ranges. It can be observed that decrease in *minWS* and increase in weight range will have positive effect on the number of WFIs.

Table III show the number of nodes generated by WFIM and SWFP-growth algorithms at different weight ranges in various databases. The *minSup* and *minWS* in T10I4D100K are set at 1% and 1%, respectively. In Twitter database, the *minSup* and *minWS* are set at 1.5% and 1.5%, respectively. In Connect database, the *minSup* and *minWS* are set at 50% and 50%, respectively. In Mushroom database, the *minSup* and *minWS* are set at 15% and 15%, respectively. It can be observed that increase in weight range has increased the number of nodes generated by both algorithms. however,

TABLE III: Number of nodes visited

Dataset	Weight Range	WFIM	Proposed
T10I4D100K	0.1 - 0.5	78,149	21,784
	0.5 - 1	932,185	125,846
	1 - 2	1,133,694	562,483
	2 - 4	3,516,791	1,373,642
	4 - 6	3,585,909	1,467,310
Twitter	0.1 - 0.5	372,151	80,214
	0.5 - 1	741,395	475,182
	1 - 2	1,245,217	845,124
	2 - 4	2,452,166	1,324,857
	4 - 6	3,451,248	2,684,751
Connect	0.1 - 0.5	942,412	751,481
	0.5 - 1	2,415,780	962,340
	1 - 2	3,614,278	1,247,527
	2 - 4	5,248,167	3,217,544
	4 - 6	6,190,652	4,581,961
Mushroom	0.1 - 0.5	271,582	63,634
	0.5 - 1	835,227	101,606
	1 - 2	2,848,660	1,783,272
	2 - 4	3,102,708	2,158,959
	4 - 6	3,226,296	2,384,382

the proposed algorithm has generated less number nodes. The reason is as follows: *the increase in weight range actually increases the W_{max} value, thereby enabling many uninteresting itemsets to be candidate weighted frequent items*. As SWFP-growth is not influenced by the W_{max}, it has not generated too many nodes.

Fig. 8(a)-(d) show the runtime consumed by WFIM and SWFP-growth algorithms on T10I4D100K, Twitter, Connect and Mushroom databases, respectively. It can be observed that although the runtime for both the algorithms increases with the increase in weight range, SWFP-growth takes less time than the WFIM. Moreover, SWFP-growth is more efficient at high weight ranges. (In general, higher weight ranges are used to find WFIs in sparse databases).

Figures 9(a)-(d) show the memory consumed by WFIM and SWFP-growth algorithms on T10I4D100K, Twitter, Connect and Mushroom databases, respectively. Similar observations as that of the runtime can be drawn for the memory.

Figures 10 (a) and (b) show the runtime requirements of PWFIM and PWFP-growth algorithms with the number of machines. It can be observed that increase in number of machines decreases the runtime for both PWFIM and PWFP-growth algorithms. However, PWFP-growth algorithm was at least 50% faster than the PWFIM algorithm.

Figures 11 (a) and (b) show the amount of data shuffled among the machines. It can be observed that increase in machines increases the amount of data shuffled for both naive and proposed algorithms. However, it can be observed that the data shuffled is very less in PWFP-growth algorithm.

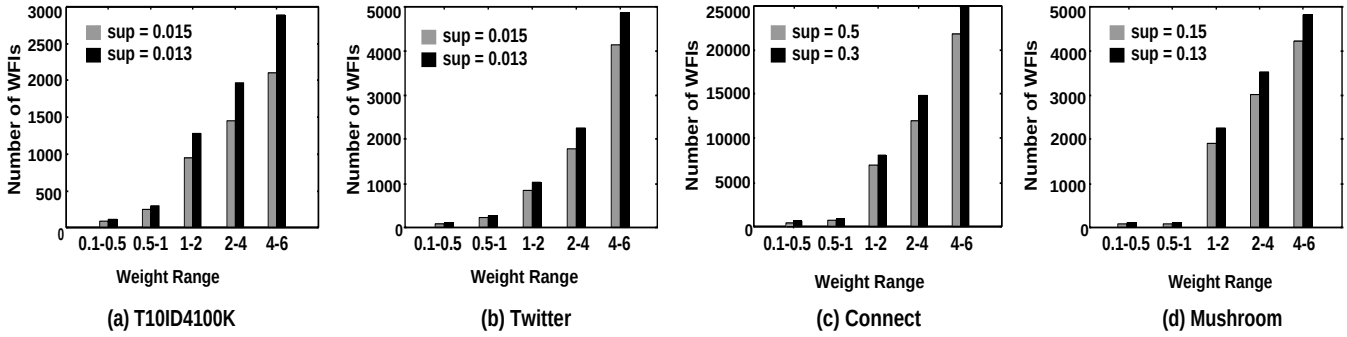


Fig. 7: WFIs generated in various databases at different *minWS* and weight ranges

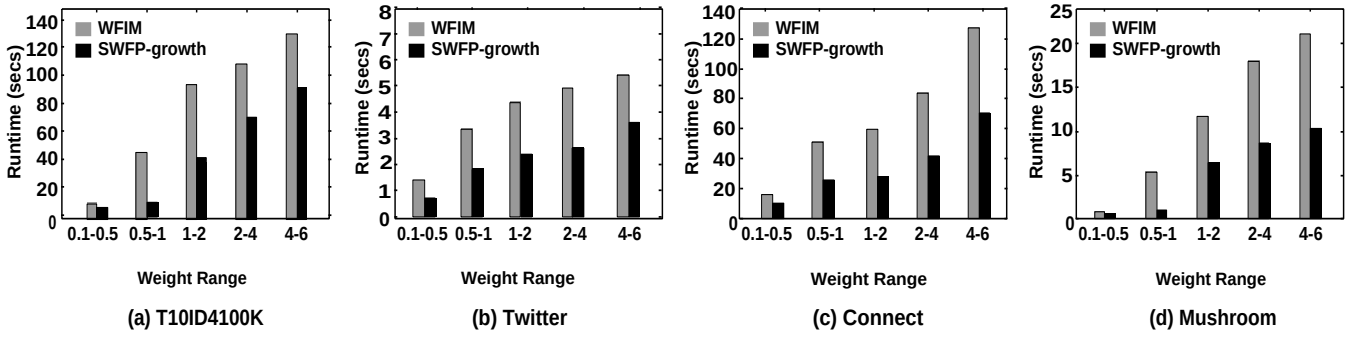


Fig. 8: Runtime comparison of WFIM and SWFP-growth algorithms on various databases

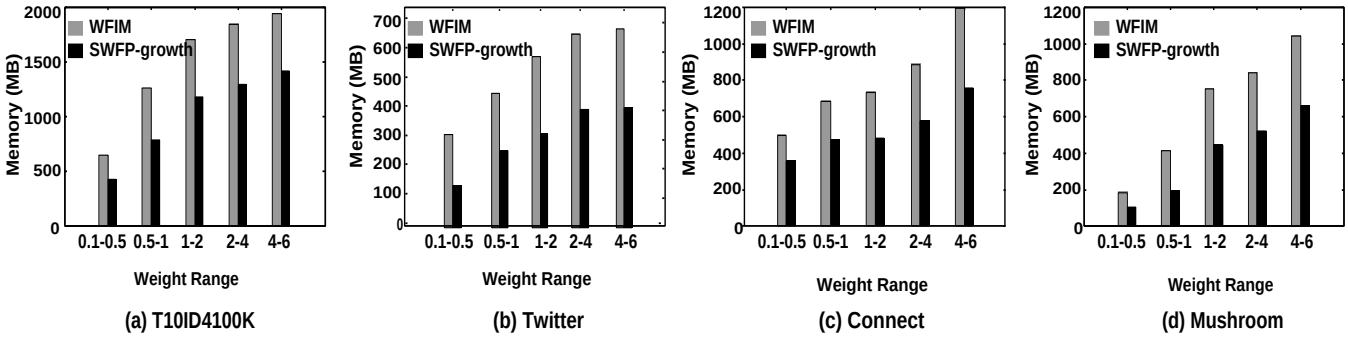
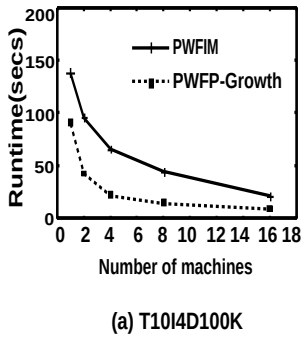


Fig. 9: Memory comparison of WFIM and SWFP-growth algorithms on various databases



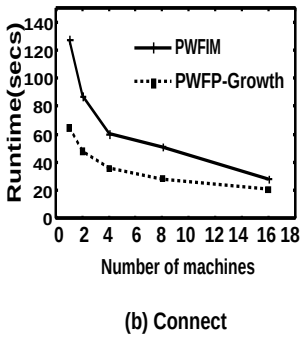
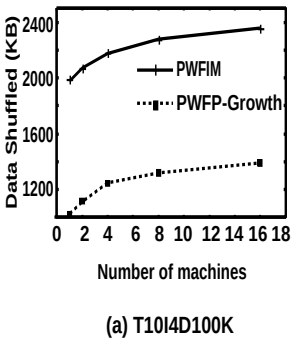


Fig. 10: Runtime comparison



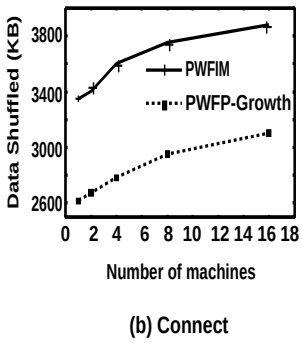
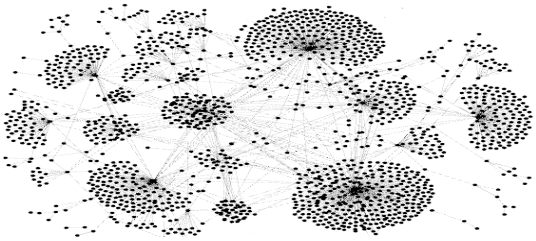


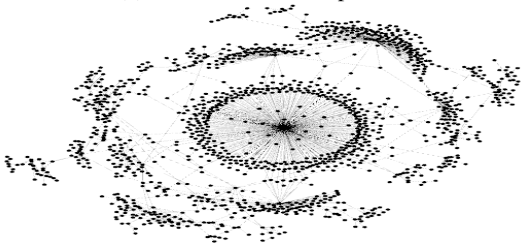
Fig. 11: Data shuffle

A. A case study on Twitter data

In this subsection, we demonstrate the usefulness of WFIs using Twitter database. The weights to items were set using TF-IDF weighting scheme. The *minWS* is set at 1.5%. The weight range is (2,4). The *minWt* is set at 1.3. Some of the interesting WFIs discovered from the database are shown in Table IV. This information regarding the WFIs can be used



(a) Manchester and explosion



(b) terrorist and attack

Fig. 12: Topology of tweets containing weighted frequent 2-itemsets

for various purposes, such as understanding the topology and event summarization. Figure 1 shows the topology of tweets containing the words 'explosion' and 'injured.' Figure 14 (a) and (b) show the topology of tweets containing the words 'Manchester and explosion' and 'Terrorist and attack,' respectively. It can be observed that their is one big cluster for the tweets containing the words 'terrorist' and 'attack,' while there were many small clusters for the words 'Manchester' and 'explosion.'

TABLE IV: Some of the interesting itemsets generated in Twitter data

Itemsets	*Sup*	*W*
{Manchester,explosion}	4849	1.8
{terrorist,attack}	2182	1.4
{explosion,injured}	1852	1.75

VII. Conclusions and Future Work

The problem of finding WFIs has been widely studied in the past. Recently, the practical applications of this model are gaining popularity in many real-world applications, such as astronomy and market-basket analytics. The popular adoption and successful industrial application of the model has been hindered by its huge computational requirements. With this motivation, this paper revisited the problem of finding WFIs and showed that the itemsets generated using the weighted average function satisfy the sorted closure property. We proposed three pruning techniques, cutoff weight, conditional pattern base elimination and pattern-growth termination, to reduce the uninteresting itemsets that have to be take into account for finding the WFIs. Two pattern-growth algorithms, SWFP-growth and PWFP-growth, have also been discussed to find the WFIs effectively. Experimental results on both synthetic and real-world databases demonstrate that proposed algorithms are runtime and memory efficient, and highly scalable as well.

In this paper, our study has been confined to finding WFIs in transactional databases. As a part of future work, we would like to extend the proposed pruning techniques to utility itemset mining and finding WFIs in uncertain databases and data streams.ad

References

[1] R. Agrawal, T. Imieliński, and A. Swami, "Mining association rules between sets of items in large databases," in *Acm sigmod record*, vol. 22, no. 2. ACM, 1993, pp. 207–216.

[2] R. Agrawal, R. Srikant *et al.*, "Fast algorithms for mining association rules," in *Proc. 20th int. conf. very large data bases, VLDB*, vol. 1215, 1994, pp. 487–499.

[3] B. Liu, W. Hsu, and Y. Ma, "Mining association rules with multiple minimum supports," in *Proceedings of the fifth ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 1999, pp. 337–341.

[4] C. H. Cai, A. W.-C. Fu, C. Cheng, and W. Kwong, "Mining association rules with weighted items," in *Database Engineering and Applications Symposium, 1998. Proceedings. IDEAS'98. International*. IEEE, 1998, pp. 68–77.

[5] U. Yun and J. J. Leggett, "Wfim: weighted frequent itemset mining with a weight range and a minimum weight," in *Proceedings of the 2005 SIAM International Conference on Data Mining*. SIAM, 2005, pp. 636–640.

[6] J.-H. Cai, X.-J. Zhao, S.-W. Sun, J.-F. Zhang, and H.-F. Yang, "Stellar spectra association rule mining method based on the weighted frequent pattern tree," *Research in Astronomy and Astrophysics*, vol. 13, no. 3, p. 334, 2013.

[7] G. Rattanaritnont, M. Toyoda, and M. Kitsuregawa, "Analyzing patterns of information cascades based on users' influence and posting behaviors," in *Proceedings of the 2nd Temporal Web Analytics Workshop*. ACM, 2012, pp. 1–8.

[8] S.-A. Bahrainian and A. Dengel, "Sentiment analysis and summarization of twitter data," in *Computational Science and Engineering (CSE), 2013 IEEE 16th International Conference on*. IEEE, 2013, pp. 227–234.

[9] J. Han, J. Pei, and Y. Yin, "Mining frequent patterns without candidate generation," in *ACM sigmod record*, vol. 29, no. 2. ACM, 2000, pp. 1–12.

[10] J. Pei and J. Han, "Constrained frequent pattern mining: a pattern-growth view," *ACM SIGKDD Explorations Newsletter*, vol. 4, no. 1, pp. 31–39, 2002.

[11] H. Li, Y. Wang, D. Zhang, M. Zhang, and E. Y. Chang, "Pfp: parallel fp-growth for query recommendation," in *Proceedings of the 2008 ACM conference on Recommender systems*. ACM, 2008, pp. 107–114.

[12] Y. Xun, J. Zhang, and X. Qin, "Fidoop: Parallel mining of frequent itemsets using mapreduce," *IEEE transactions on Systems, Man, and Cybernetics: systems*, vol. 46, no. 3, pp. 313–325, 2016.

[13] F. Tao, F. Murtagh, and M. Farid, "Weighted association rule mining using weighted support and significance framework," in *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2003, pp. 661–666.

[14] B. Vo, F. Coenen, and B. Le, "A new method for mining frequent weighted itemsets based on wit-trees," *Expert Systems with Applications*, vol. 40, no. 4, pp. 1256–1264, 2013.

[15] J. C.-W. Lin, W. Gan, P. Fournier-Viger, H.-C. Chao, and T.-P. Hong, "Efficiently mining frequent itemsets with weight and recency constraints," *Applied Intelligence*, pp. 1–24, 2017.

[16] J. C.-W. Lin, W. Gan, P. Fournier-Viger, T.-P. Hong, and V. S. Tseng, "Weighted frequent itemset mining over uncertain databases," *Applied Intelligence*, vol. 44, no. 1, pp. 232–250, 2016.

[17] C. Ahmed, S. Tanbeer, B.-S. Jeong, and Y.-K. Lee, "Mining weighted frequent patterns using adaptive weights," *Intelligent Data Engineering and Automated Learning–IDEAL 2008*, pp. 258–265, 2008.

[18] E. Baralis, L. Cagliero, P. Garza, and L. Grimaudo, "Pawi: parallel weighted itemset mining by means of mapreduce," in *Big Data (BigData Congress), 2015 IEEE International Congress on*. IEEE, 2015, pp. 25–32.