








Mining Sequential Patterns in Uncertain Databases Using Hierarchical Index Structure

Kashob Kumar Roy¹ , Md Hasibul Haque Moon¹ , Md Mahmudur Rahman¹ , Chowdhury Farhan Ahmed¹  , and Carson K. Leung² 

¹ Department of Computer Science & Engineering, University of Dhaka, Bangladesh

² Department of Computer Science, University of Manitoba, Canada

kashobroy@gmail.com, hasibulhq.moon@gmail.com, mahmudur@du.ac.bd,
farhan@du.ac.bd , kleung@cs.umanitoba.ca

Abstract. In this uncertain world, data uncertainty is inherent in many applications and its importance is growing drastically due to the rapid development of modern technologies. Nowadays, researchers have paid more attention to mine patterns in uncertain databases. A few recent works attempt to mine frequent uncertain sequential patterns. Despite their success, they are incompetent to reduce the number of false-positive pattern generation in their mining process and maintain the patterns efficiently. In this paper, we propose multiple theoretically tightened pruning upper bounds that remarkably reduce the mining space. A novel hierarchical structure is introduced to maintain the patterns in a space-efficient way. Afterward, we develop a versatile framework for mining uncertain sequential patterns that can effectively handle weight constraints as well. Besides, with the advent of incremental uncertain databases, existing works are not scalable. There exist several incremental sequential pattern mining algorithms, but they are limited to mine in precise databases. Therefore, we propose a new technique to adapt our framework to mine patterns when the database is incremental. Finally, we conduct extensive experiments on several real-life datasets and show the efficacy of our framework in different applications.

Keywords: Sequential Pattern Mining · Uncertain Database · Weighted Sequential Patterns · Incremental Database.

1 Introduction

Sequential Pattern Mining is an important and challenging data mining problem [11,13] with broad applications where the order of the itemsets or events in a sequence is important. There are many applications such as environmental surveillance, medical diagnosis, security, and manufacturing systems etc where uncertainty is inherent in nature due to several limitations: (i) our limited understanding of reality; (ii) limitations of the observation equipment; or (iii) limitations of available resources for the analysis of data, etc. A large number of approaches have been introduced in [1,5,7,8] to mine frequent itemsets from uncertain databases. Algorithms proposed in [3,15] mine sequential patterns in

uncertain databases. However, in the real world, not all items are equally important. For example, in biomedical data analysis, some genes are more vital than others in causing a particular disease. Weighted pattern mining methods are proposed in [6,14] for this task. Rahman et al. [12] handle weight constraints in mining uncertain sequential patterns by maintaining weight and expected support threshold separately. Thus, it can efficiently mine sequences having high frequencies with high weights but incompetent to mine sequences which have low frequencies with high weights or high frequencies with low weights. Besides, existing uncertain sequential pattern mining methods have some vital limitations such as: a) generation of a huge number of false-positive patterns due to the pruning upper bounds; b) inefficient maintenance of candidate patterns, which results in costly support computation; and c) lack of a sophisticated weight upper bound to mine weighted patterns efficiently while maintaining anti-monotone property. To address these limitations, we propose multiple novel pruning upper bounds that are theoretically tightened than respective upper bounds already introduced in the literature and utilize a hierarchical index structure to maintain potential candidate patterns in a space-efficient way.

Moreover, with the advent of modern technologies, most databases are dynamic and incremental in nature. A large number of researches [2,4,9] have been successful in incremental pattern mining. But none of the existing uncertain sequential pattern mining algorithms are effective in handling the dynamic nature because running batch algorithms from scratch after each increment is not a feasible solution in the sense of time. To the best of our knowledge, our proposed technique is the first work to mine sequential patterns in incremental uncertain databases. In summary, our contributions in this work are as follows,

1. Three theoretically tightened upper bounds: $expSup^{cap}$, wgt^{cap} , $wExpSup^{cap}$ to reduce the search space of mining potential candidate patterns.
2. A novel hierarchical index structure, *USeq-Trie*, to maintain the patterns.
3. A faster method, *SupCalc*, to compute expected support of patterns.
4. An efficient algorithm, *FUSP*, to mine sequential patterns in uncertain database.
5. An approach *InUSP* for incremental mining of uncertain sequential patterns.

Extensive experimental analysis validates the efficacy of our proposed methods and shows that our methods consistently outperform other baseline approaches.

2 Background Study

Related Works. Among a plethora of research on sequential pattern mining, *GSP* [13] works based on candidate generation and testing paradigm whereas *PrefixSpan* [11] follows the divide-and-conquer approach to mine frequent sequences in precise databases. *PrefixSpan* [11] expands patterns by recursively projecting the database into smaller parts and mining local patterns in those prefix-projected databases. Uncertain data has gained great attention in re-

Table 1: Initial Database, *DB*

Id	Uncertain Sequence
1	(a:0.9, c:0.6)(a:0.7)(b:0.3)(d:0.7)
2	(a:0.6, c:0.4)(a:0.5)(a:0.4, b:0.3)
3	(a:0.3)(a:0.2, b:0.2)(a:0.4, b:0.3, g:0.5)
4	(a:0.1, c:0.1)(a:0.3, b:0.1, c:0.4)
5	(d:0.1)(a:0.4)(d:0.1)(a:0.5, c:0.6)
6	(b:0.3)(b:0.4)(a:0.1)(a:0.1, b:0.2)

Table 2: Weight Table

Item	Weight	Item	Weight
a	0.8	b	1.0
c	0.9	d	0.9
e	0.7	f	0.9
g	0.8		

cent years [1,6,10,12,15]. Inspired by *PrefixSpan*, *U-PrefixSpan* [10] mines probabilistic frequent sequences whereas *uWSequence* [12] mines expected support-based frequent sequences with weight constraints in uncertain databases. *uWSequence* [12] uses $expSupport^{top}$ upper-bound to prune the mining space of patterns. They use weight threshold as an extra level of filtering which is not aligned with the concept of weighted support defined in [14] for precise databases. Following [14], we introduce the concept of weighted expected support in uncertain sequential pattern mining that considers both expected support and weight of patterns simultaneously. Further, researchers proposed various algorithms in [2,4,9] to handle increments in databases. *IncSpan* [2] introduces the concept of buffering semi-frequent sequences (*SFS*) mined from initial databases which may become frequent after future increments. *WIncSpan* [4] finds weighted sequential patterns in incremental precise databases. Despite the promising significance of incremental uncertain sequential pattern mining in different applications, existing works are not capable to mine patterns efficiently. Hence, we introduce a new concept of promising frequent sequences (*PFS*) to improve the efficiency.

Preliminaries. Let $I = \{i_1, i_2, \dots, i_n\}$ be the set of all items in a database. An event $e_i = (i_1, i_2, \dots, i_k)$ is a subset of I . A sequence is an ordered set of events. For example, $\alpha = \langle (i_2), (i_1, i_5), (i_1) \rangle$ consists of 3 consecutive events. In uncertain sequences, items in each event are assigned with their existential probabilities such as $\alpha = \langle (i_2: P_{i_2}), (i_1: P_{i_1}, i_5: P_{i_5}), (i_1: P_{i_1}) \rangle$. An uncertain sequential database is a collection of uncertain sequences shown in Table 1. Support of a sequence α in a database is the number of data tuples that contain α as a subsequence. In this paper, we follow the definition of expected support (*expSup*) for a sequence (items within the sequence are independent) which is defined in [12] as the sum of the maximum possible probabilities of that sequence in each data tuple where the probability of a sequence is computed simply by multiplying the uncertainty value of its all items. A sequence α can be extended with an item i in two ways: i) *i-extension*, insert i to the last event of α , and ii) *s-extension*, add i to α as a new event. Weight of a sequence (*sWeight*) is the sum of its each individual item's weight divided by the length of the sequence [14] i.e., the total number of items in the sequence. According to Table 1 and Table 2, for sequence $\alpha = \langle (a)(b) \rangle$, support of α is 5, $expSup(\alpha) = \max(0.9 \times 0.3, 0.7 \times 0.3) + \max(0.6 \times 0.3, 0.5 \times 0.3) + \max(0.3 \times 0.2, 0.3 \times 0.3, 0.2 \times 0.3) + (0.1 \times 0.1) + 0 + (0.1 \times 0.2) = 0.57$, and $sWeight(\alpha) = (0.8 + 1.0)/2 = 0.9$ as per the definitions.

3 A Framework for mining Uncertain Sequential Patterns

In this section, we propose a new framework for mining sequential patterns in uncertain databases efficiently with/without the weight constraints in mining patterns followed by discussing the incremental mining approach when the database would be of dynamic nature.

Definitions. $maxPr$ is the maximum possible probability of a sequence $\alpha = \langle (i_1)(i_2)\dots(i_{|\alpha|}) \rangle$ in the whole database [12],

$$maxPr(\alpha) = \prod_{k=1}^{|\alpha|} (\hat{P}_{DB|\alpha_{k-1}}(i_k)) \text{ where } \alpha_{k-1} = \langle (i_1)\dots(i_{k-1}) \rangle \quad (1)$$

where $\hat{P}_{DB|\alpha}(i)$ = maximum possible probability of item i in a database $DB | \alpha$ that is the projection of original database with α as current prefix [11]. Moreover, [12] shows that the $maxPr$ measure holds anti-monotone property. Similar to $maxPr$, we define another measure $maxPr_S(\alpha)$ as the maximum probability of a pattern α in a single data sequence S . According to Table 1, the $maxPr(\langle (c)(a) \rangle) = 0.6 \times 0.7 = 0.54$ and $maxPr(\langle (ac) \rangle) = 0.9 \times 0.6 = 0.54$; where for the 1st data sequence, $maxPr_S(\langle (a)(b) \rangle) = max(0.9 \times 0.3, 0.7 \times 0.3) = 0.27$. We define an upper bound of expected support of a sequence α of length m as,

$$expSup^{cap}(\alpha_m) = maxPr(\alpha_{m-1}) \times \sum_{\forall S \in (DB|\alpha_{m-1})} maxPr_S(i_m) \quad (2)$$

Lemma 1. For a sequence α , $expSup^{cap}(\alpha) \geq expSup(\alpha)$ and $expSup(\alpha) \geq expSup(\alpha')$, where $\alpha \subseteq \alpha'$; $\therefore expSup^{cap}(\alpha) \geq expSup(\alpha')$. If $expSup^{cap}(\alpha) < a$ minimum threshold γ holds, then $expSup(\alpha) < \gamma$ and $expSup(\alpha') < \gamma, \forall \alpha \supseteq \alpha'$ must be true. Thus it satisfies the anti-monotonicity constraints.

Lemma 2. For a sequence α , $expSup^{cap}(\alpha) \leq expSupport^{top}(\alpha)^3$ always holds. Hence, $expSup^{cap}(\alpha)$ significantly reduces the search space in mining patterns and leads to a smaller number of false positive patterns than $expSupport^{top}(\alpha)$.

Later on, we define few more definitions where each item has a weight to indicate its importance. We will be consistent with weighted pattern mining in following sections. Note that our framework is easily adaptable to mine patterns without weight constraints that is discussed in the experiments section. Following the concept of *weighted support* for precise database in [14], we define *weighted expected support* of a sequence α as $WES(\alpha) = expSup(\alpha) \times sWeight(\alpha)$. According to Tables 1 and 2, $WES(\langle (a)(b) \rangle) = 0.57 \times 0.9 = 0.513$. A sequence α is called *weighted sequential pattern* if $WES(\alpha)$ meets a minimum threshold. This threshold is defined to be $minWES = min_sup \times (size\ of\ the\ whole\ database) \times WAM \times wgtFct$. Here, min_sup is user given value in range $[0,1]$ related to a sequence's frequency, WAM is weighted arithmetic mean of all item-weights

³ uWSequence[12] defines the upper bound of expected support as $expSupport^{top}(\alpha) = maxPr(\alpha_{m-1}) \times maxPr(i_m) \times sup_{i_m}$ where sup_{i_m} is the support count of i_m .

present in the database and defined as $WAM = (\sum_{i \in I} w_i \times f_i) / \sum_{i \in I} f_i$, where w_i and f_i are the weight and frequency of item i in current database. Hence, the value of WAM changes after each increment in the database. $wgtFct$ is a user-given positive value chosen to tune the mining of weighted sequential patterns. Choice of min_sup and $wgtFct$ depends on how much frequent and weighted patterns are required in the respective applications.

However, the measure WES does not hold anti-monotone property as any item with higher weight can be appended to a weighted-infrequent sequence and the resulting super-sequence may become weighted-frequent. So, to employ anti-monotone property in mining weighted frequent patterns, we propose two other upper bound measures, wgt^{cap} and $wExpSup^{cap}$, which are used as upper bound of *weight* and *weighted expected support* respectively. Upper bound of weight of a sequence α , $wgt^{cap}(\alpha)$ is defined as,

$$wgt^{cap}(\alpha) = \max(mxW_{DB}(DB|\alpha), mxW_s(\alpha)) \quad (3)$$

where $mxW_{DB}(DB|\alpha)$ is the *maximum weight of all frequent items in the α -projected database* and $mxW_s(\alpha)$ is the *maximum weight of all items in the sequence α* . To enforce the anti-monotone property of weighted frequent patterns in precise databases, authors in [4,14] make an attempt to use the maximal weight of all items in database as upper bound of weight of a sequence. It is obvious to see that wgt^{cap} of a sequence is always less than or equal to the maximal weight of all items in database. As wgt^{cap} becomes tighter, it generates fewer false positive patterns compared to the existing methods.

Lemma 3. *For any sequence α , $wgt^{cap}(\alpha)$ is at least equal to the $sWeight$ value of α and all of its supersequences, α' . Because, $wgt^{cap}(\alpha) \geq sWeight(\alpha)$ and $wgt^{cap}(\alpha) \geq wgt^{cap}(\alpha')$, where $\alpha \subseteq \alpha'$; $\therefore wgt^{cap}(\alpha) \geq sWeight(\alpha')$.*

The proposed upper bound of weighted expected support is defined as,

$$wExpSup^{cap}(\alpha) = expSup^{cap}(\alpha) \times wgt^{cap}(\alpha) \quad (4)$$

Lemma 4. *For a sequence α , if $wExpSup^{cap}(\alpha) < minWES$, then none of α and its supersequences can be weighted frequent. Because, $wExpSup^{cap}(\alpha) \geq WES(\alpha)$, and $wExpSup^{cap}(\alpha) \geq WES(\alpha')$, for all $\alpha \subseteq \alpha'$.*

According to Lemma 4, we can safely define our pruning condition to reduce the search space of patterns in pattern-growth based mining as follows:

If for any k -sequence α , $wExpSup^{cap}(\alpha) < minWES$, then searching possible extension of α to $(k+1)$ -sequence can be pruned, i.e., neither α nor any super sequences of α would be frequent at all.

Moreover, Lemma 4 ensures that our proposed algorithms do not generate any false negative patterns. However, as $wExpSup^{cap}(\alpha) \geq WES(\alpha)$, some patterns may be discovered with $wExpSup^{cap}(\alpha) \geq minWES$ but $WES(\alpha) < minWES$. An extra scan of the database is required to remove them. We have omitted proof of the lemmas due to space limitation.

3.1 USeq-Trie: Maintenance of Patterns

We use a hierarchical data structure, named as *USeq-Trie*, to store uncertain sequences and update their weighted expected support efficiently. Each node in the *USeq-Trie* represents an item in a sequence and will be created as either *s-extension* or *i-extension* from its parent node. Recall that a sequence is an ordered set of events, and an event is a set of items. In *s-extension*, the edge label is added as a different event. In *i-extension*, it is added in the same event as its parent. Each edge is labeled by an item. The edge labels in a path to from root to a node forms a pattern. For example, $\langle(a)\rangle$, $\langle(b)\rangle$, $\langle(ab)\rangle$, $\langle(c)\rangle$, $\langle(b)(c)\rangle$, $\langle(d)\rangle$, $\langle(cd)\rangle$ and $\langle(c)(d)\rangle$ are sequential patterns who are stored into *USeq-Trie* shown in Fig. 1. In this figure, the *s-extensions* are denoted by the *solid lines* and *i-extensions* by *dashed lines*. For simplicity of the figure, we are not showing edge labels here. Each node represents a (weighted) frequent uncertain sequence and stores its (weighted) expected support. Now, we present an efficient method, *SupCalc*, to calculate *expSup* or *WES* for each candidate pattern stored in a *USeq-Trie*.

Support Calculation, SupCalc. It reads sequences from the dataset one by one and updates the support of all patterns in *USeq-Trie* against them. For a sequence $\alpha = \langle e_1 e_2 \dots e_n \rangle$ (where e_i is an event/itemset), the steps are following,

1. Define an array of size n at each node. For the root node, all values are 1.0. At a particular node, the maximum expected support of pattern s from root to that node is stored at proper indices of the node's array - are the ending positions of s as a sub-sequence in α . The values at other indices are 0.0.
2. While traversing the *USeq-Trie* in depth-first order: (i) For a node created by a *s-extension* with an item i_k , we iterate over all events in α and calculate the support of the current pattern s (ends with i_k in a new event) by multiplying the probability of item i_k in current event e_m with the maximum probability in the parent node's array up to the event e_{m-1} . The resulting support is stored at position m in the following node's array. (ii) For *i-extension*, the

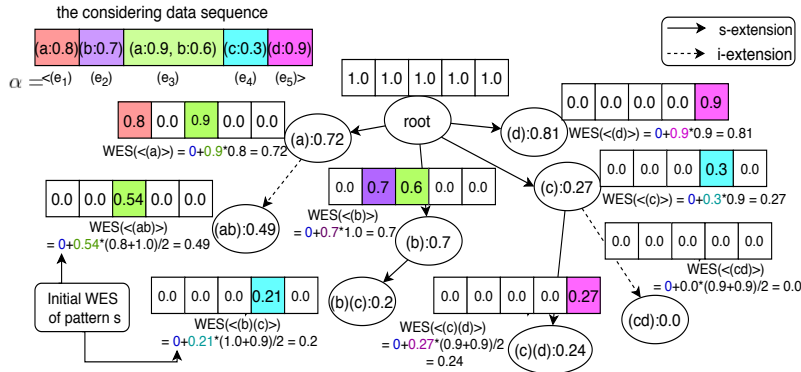


Fig. 1: An efficient way to compute *WES* of patterns stored into *USeq-Trie*

support will be calculated by multiplying the probability of the item i_k in e_m with the value at position m in the parent node's array and stored at position m in the following child node's array. After that, the maximum value in the resulting array multiplied by its weight will be added to the weighted expected support of the current pattern at the corresponding node.

3. Use the resultant array to calculate the weighted expected support of all super patterns while traversing the next child nodes.

Fig. 1 shows the resulting *USeq-Trie* after updating *WES* for all the stored patterns against a sequence, $\alpha = \langle (a:0.8)(b:0.7)(a:0.9, b:0.6)(c:0.3)(d:0.9) \rangle$.

Complexity of *SupCalc*. It takes $O(N \times |\alpha|)$ for updating N number of nodes against the sequence α . Therefore, the total time complexity of actual support calculation is $O(|DB| \times N \times k)$ where k is the maximum sequence length in the dataset. It outperforms the procedure used in *uWSequence* [12] which needs $O(|DB| \times N \times k^2)$ to calculate a sequence's actual expected support. Moreover, we can remove false-positive patterns and find frequent ones from the *USeq-Trie* in $O(N)$. Thus, the use of *USeq-Trie* has made our method efficient.

3.2 FUSP: Faster mining of Uncertain Sequential Patterns

Inspired by *PrefixSpan* [11], we propose *FUSP* to mine weighted sequential patterns in an uncertain database. It uses the $wExpSup^{cap}$ measure and *SupCalc* method to reduce the search space and improve the efficiency. The sketch of *FUSP* algorithm is as follows.

1. Process the database such that the existential probability of an item in a sequence is replaced with the maximum probability of all of its next occurrences in this sequence. This idea is similar to the *preprocess* function of *uWSequence* [12]. This preprocessed database will be used to run the *PrefixSpan*-like mining approach to find the candidates for frequent sequences. While processing, sort the items in an event/itemset in lexicographical order.
2. Calculate *WAM* of all items present in the current database and calculate the threshold of weighted expected support, *minWES*.
3. Find length-1 frequent items and for each item, project the preprocessed database into smaller parts and expand longer patterns recursively. Store the candidates into a *USeq-Trie*.
4. While growing longer patterns, extend current prefix α to α' with an item β as *s-extension* or *i-extension* according to the pruning condition.
5. Use of $wExpSup^{cap}$ value instead of actual support generates few false-positive candidates. Scan the whole actual database, update weighted expected supports and prune false-positive candidates based on their *WES*.

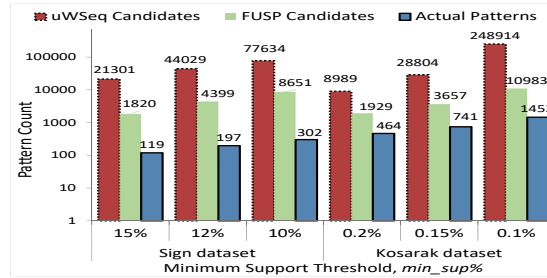
3.3 InUSP: Incremental mining of Uncertain Sequential Patterns

Existing incremental works [2,4] follow the technique to lower the minimum support threshold by a user-given buffer ratio, $\mu \in [0, 1]$, and find *almost frequent*

sequence called *SFS* - stating that most of the frequent patterns in the appended database will either come from *SFS* or already frequent sequences (*FS*) in the initial database. Inspired by this concept, we use $minWES' = minWES \times \mu$ to find *SFS* where $minWES' \leq WES < minWES$, along with *FS* where $WES \geq minWES$. However, we argue that *SFS* is not necessarily enough to capture new frequent patterns in future increments. Let us consider some cases: (a) an increment to the database may introduce a new sequence which was initially absent in both *FS* and *SFS* but frequently appeared in later increments; (b) a sequence had become infrequent after an increment but could have become semi-frequent or even frequent again after next few increments. There are many real-life cases where new frequent patterns might appear in future increments due to its seasonal behavior or different other characteristics. Existing approaches do not handle these cases. To address these cases, we propose to maintain another set of sequences denoted as *Promising Frequent Sequences (PFS)* which are neither globally frequent nor semi-frequent after each increment ΔDB introduced into *DB* but their *WES* satisfy a user-specified threshold that can be defined as $LWES = \gamma \times \mu \times min_sup \times |\Delta DB| \times WAM \times wgtFct$ where γ is a constant factor, to find locally frequent patterns in ΔDB at a particular point. Here, the globally frequent or semi-frequent implies when considering the size of the entire database, and locally frequent when using the size of only one increment. Intuitively, we can say that locally frequent patterns may become globally frequent or semi-frequent after next few increments. The patterns whose *WES* values do not meet the local threshold *LWES*, are very unlikely to become globally frequent or semi-frequent. Thus maintaining *PFS* may significantly increase the performance of an algorithm in finding the almost complete set of frequent patterns after each increment. Therefore, we devise *InUSP* to incorporate the concept of *PFS* in mining patterns. Instead of performing *FUSP* from scratch after each increment, *InUSP* works only on ΔDB . Initially, it runs *FUSP* once to find out *FS* and *SFS* from initial database and uses *USeq-Trie* to store *FS* and *SFS*. In addition, a different *USeq-Trie*, which is initially empty, is used to store *PFS*.

After each increment ΔDB , the steps of *InUSP* algorithm are as follows:

1. Update the values of *database size*, *WAM*, *minWES*, and *minWES'*.
2. Run *FUWS* only in ΔDB to find locally frequent sequences (*LFS*) against a local threshold, *LWES*, and store them into *USeq-Trie*. Users can choose *LWES* based on the aspects of application.
3. For all α in *FS*, *SFS* and *PFS*, update WES_α using the *SupCalc* method.
 - if $WES_\alpha < LWES$, delete α 's information.
 - else if $WES_\alpha < minWES'$, move α to *PFS'*.
 - else if $WES_\alpha < minWES$, move α to *SFS'*.
 - else move α to *FS'*.
4. Move new patterns α from *LFS* to *PFS'* or *SFS'* or *FS'* based on WES_α .
5. Use *FS'*, *SFS'*, and *PFS'* as *FS*, *SFS*, and *PFS* respectively for the next increment.

Fig. 2: *FUSP* outperforms *uWSequence* in candidate generationTable 3: Runtime (seconds) comparison between *uWSequence* and *FUSP*

Sign Dataset			Kosarak Dataset			Fifa Dataset		
min_sup	uWSeq.	FUSP	min_sup	uWSeq.	FUSP	min_sup	uWSeq.	FUSP
20%	717.69	10.64	0.25%	5942.06	348.32	20%	1615.50	12.73
18%	1116.75	18.34	0.22%	7102.27	443.13	18%	2943.45	25.85
15%	2052.04	32.64	0.2%	8581.56	475.12	17%	4003.97	34.79
12%	4316.43	72.39	0.18%	14622.38	659.30	16%	6114.34	56.05
10%	7275.41	122.94	0.15%	33864.18	1029.70	15%	9033.86	74.95

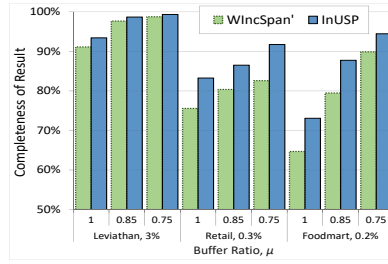
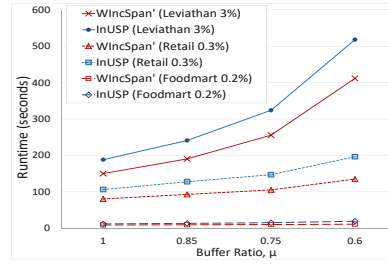
4 Experimental Results

We have evaluated our algorithms using several real-life and popular datasets such as *Sign*, *Kosarak*, *Fifa*, *Leviathan*, *Retail*, *Foodmart*, *Chainstore*, and *Online Retail* from *SPMF*⁴ data repository. We assigned probability and weight values to the items of these datasets as all of them were precise and none of them contained weight information. We followed normal distribution with *mean* of 0.5 and *standard deviation* of 0.25 (for probabilities) or 0.125 (for weights) to generate these values. We implemented our algorithms in *Python* programming language and a machine with *Core™ i5-9600U 2.90GHz CPU* and *8GB RAM*.

Performance of *FUSP*. We have compared with the recent algorithm, *uWSequence* [12], which proposed a framework where the definition of weighted sequential pattern in uncertain databases is different from ours. Furthermore, *uWSequence* [12] outperforms existing methods for mining sequential patterns also without weight constraints in uncertain databases. So, to show the efficiency of *FUSP* in mining uncertain sequential patterns without weight constraints, we have compared *FUSP* with the current best *uWSequence* by setting the weights of all items to 1.0 which brings both algorithms under a unifying framework.

(a) False Candidate Generation: Recall that both *FUSP* and *uWSequence* work like *PrefixSpan* using some upper bound of actual expected support value and thus, generate some false positive candidates. From Fig. 2, we can see that

⁴ <http://www.philippe-fournier-viger.com/spmf/index.php?link=datasets.php>

Fig. 3: Completeness comparison between *WIncSpan'* and *InUSP*Fig. 4: Runtime comparison between *WIncSpan'* and proposed *InUSP*

FUSP generates a smaller number of false candidates for any support threshold as it uses a tighter upper bound. For example, in the *Sign* (dense) dataset with 15% minimum support threshold, it generates 11 times fewer candidates compared to *uWSequence*. In *Kosarak* (sparse) with 0.15% support threshold, *FUSP* generates only 79.7% false candidates where for *uWSequence*, it is 97.4%.

(b) Runtime Analysis: *FUSP* needs to maintain a smaller number of candidate patterns in its mining process and uses a faster method to calculate expected support of a pattern. Thus, it is a way faster than the *uWSequence* for any support threshold. Results shown in Table 3 validates this claim. We can see *FUSP* is 50-70 times faster in *Sign* dataset for different thresholds. Interestingly, the difference in their runtime increases with the decrease in the threshold parameter. We have found similar results also in other datasets.

Performance of the Incremental Technique, *InUSP*. We have modified the current best incremental solution, *WIncSpan* [4] to work in uncertain data by replacing the core PrefixSpan-like algorithm by *FUSP* so that both the proposed *InUSP* and modified *WIncSpan'* mine weighted sequential patterns from uncertain database. The baseline approach is running *FUSP* from scratch in the whole updated database after each increment. We define completeness of the result from an incremental solution to be the percentage of patterns found with respect to the result of the baseline. To use the datasets as incremental ones, we used the first 50% of the dataset to be the initial part and then introduced 5 increments of random sizes⁵, unless mentioned otherwise.

(a) Analysis with respect to buffer ratio: Buffer ratio, $\mu = 1.0$ means no buffer and lower values mean larger buffers to store semi-frequent sequences. Thus, with lower μ , incremental approaches generate and maintain more patterns which help to increase the completeness of their result. However, due to local mining in incremented portions and maintaining additional promising sequences, *InUSP* always achieves more completeness than *WIncSpan'*. For the same reason, it also requires slightly more time than *WIncSpan'*. From Fig.

⁵ For the *Retail* market-basket dataset, we used the first one-fifth transactions (1st month) as the initial portion and then 4 increments to represent the next 4 months.

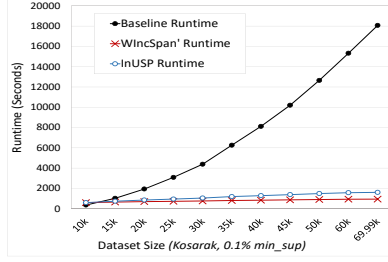


Fig. 5: Comparison of scalability using *Kosarak* dataset

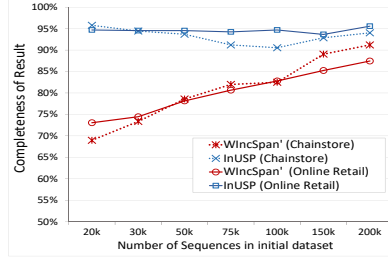


Fig. 6: Change in completeness for different initial sizes of a dataset

3 and Fig. 4, we can see the trade-off between completeness and runtime. We observe that difference in completeness is larger in datasets like *Retail* and *Foodmart* (market-basket) where increments contain frequent items or introduce new items frequently than datasets like *Leviathan* (word sequences) where the initial database contains almost all of the frequent sequences. By repeating this experiment in other datasets and by varying the support threshold, we find that though *InUSP* consumes slightly more time, it outperforms *WIncSpan'* in terms of completeness of result in every dataset for any combination of μ and min_sup .

(b) Scalability Analysis: To test scalability we have run *InUSP*, *WIncSpan'* and the baseline approach in several large datasets introducing several increments. Fig. 5 shows the result for *Kosarak* dataset with $min_sup = 0.1\%$. *InUSP* and *WIncSpan'* requires slightly more time at the initial point as they have to find and buffer the semi-frequent patterns for future use. After that, at any point of dataset increment, both of them take significantly less time to find the updated set of frequent sequences. Our proposed technique outperforms the baseline approach in terms of scalability and although it takes slightly more time than *WIncSpan'*, the difference is negligible as *InUSP* provides better completeness.

(c) Varying Initial Size of Datasets: We considered different initial sizes for this analysis and introduced required number of increments (each sized 50-80% of the initial size) to use the full dataset. Fig. 6 shows the result in *Chainstore* and *Online Retail* dataset with $min_sup = 0.05\%$ for both. We have found that the smaller the initial dataset, the more are the sequences to be found as new patterns after the increments. The completeness of incremental approaches also depends on the distribution of items among the increments. As a result, the completeness of *WIncSpan'* is competitive only if the initial dataset contains sufficient sequences compared to the total size of all future increments. However, the completeness of *InUSP* is less affected by initial size as it also mines in the incremented portions.

5 Conclusions

In this work, our proposed *FUSP* algorithm can mine sequential patterns in uncertain databases with or without weight constraints. It uses multiple theo-

retically tightened upper bounds in pruning technique and hence, generates a smaller number of false-positive patterns compared to the state-of-the-art works. Furthermore, the use of a space-efficient data structure *USeq-Trie* for pattern maintenance and an efficient method *SupCalc* for support calculation, has made *FUSP* superior to other works in terms of runtime. In case of incremental mining, the concept of promising frequent sequences lifts the effectiveness of our *InUSP* algorithm. The experimental analysis shows that our proposed techniques can be great tools for a lot of real-life applications such as medical records, sensor network, user behavior analysis, privacy-preserving data mining, that use uncertain sequential data. We hope that the concept of *USeq-Trie* structure and promising frequent sequences will help researchers to design efficient mining methods in related fields (e.g., uncertain data streams, spatio-temporal data, etc).

Acknowledgement. This project is partially supported by NSERC (Canada) and University of Manitoba.

References

1. Ahmed, A.U., Ahmed, C.F., Samiullah, M., Adnan, N., Leung, C.K.S.: Mining interesting patterns from uncertain databases. *Information Sci.* **354**, 60–85 (2016)
2. Cheng, H., Yan, X., Han, J.: IncSpan: incremental mining of sequential patterns in large database. In: ACM SIGKDD. pp. 527–532 (2004)
3. Ge, J., Xia, Y., Wang, J.: Mining uncertain sequential patterns in iterative MapReduce. In: PAKDD. pp. 243–254 (2015)
4. Ishita, S.Z., Noor, F., Ahmed, C.F.: An efficient approach for mining weighted sequential patterns in dynamic databases. In: Industrial ICDM. pp. 215–229 (2018)
5. Le, T., Vo, B., Huynh, V.N., Nguyen, N.T., Baik, S.W.: Mining top-k frequent patterns from uncertain databases. *Applied Intelligence* pp. 1–11 (2020)
6. Li, Z., Chen, F., Wu, J., Liu, Z., Liu, W.: Efficient weighted probabilistic frequent itemset mining in uncertain databases. *Expert Systems* (2020)
7. Lin, C.W., Hong, T.P.: A new mining approach for uncertain databases using cufp trees. *Expert Systems with Applications* **39**(4), 4084–4093 (2012)
8. Lin, J.C.W., Gan, W., Fournier-Viger, P., Hong, T.P., Tseng, V.S.: Weighted frequent itemset mining over uncertain databases. *Appl. Intell.* **44**(1), 232–250 (2016)
9. Lyu, X., Ma, H.: An efficient incremental mining algorithm for discovering sequential pattern in wireless sensor network environments. *Sensors* (2019)
10. Muzammal, M., Raman, R.: Mining sequential patterns from probabilistic databases. In: PAKDD. pp. 210–221 (2011)
11. Pei, J., Han, J., Mortazavi-Asl, B., Wang, J., Pinto, H., Chen, Q., Dayal, U., Hsu, M.C.: Mining sequential patterns by pattern-growth: The PrefixSpan approach. *IEEE TKDE* **16**(11), 1424–1440 (2004)
12. Rahman, M.M., Ahmed, C.F., Leung, C.K.S.: Mining weighted frequent sequences in uncertain databases. *Information Sciences* **479**, 76–100 (2019)
13. Srikant, R., Agrawal, R.: Mining sequential patterns: Generalizations and performance improvements. In: EDBT. pp. 1–17 (1996)
14. Yun, U.: A new framework for detecting weighted sequential patterns in large sequence databases. *Knowledge-Based Systems* **21**(2), 110–122 (2008)
15. Zhao, Z., Yan, D., Ng, W.: Mining probabilistically frequent sequential patterns in large uncertain databases. *IEEE TKDE* **26**(5), 1171–1184 (2013)