Mining Sequential Patterns from Probabilistic Databases*

Muhammad Muzammal** and Rajeev Raman

Department of Computer Science, University of Leicester, UK.
{mm386,r.raman}@mcs.le.ac.uk

Abstract. We consider *sequential pattern mining* in situations where there is uncertainty about which *source* an *event* is associated with. We model this in the *probabilistic database* framework and consider the problem of enumerating all sequences whose *expected* support is sufficiently large. Unlike frequent itemset mining in probabilistic databases [C. Aggarwal et al. KDD'09; Chui et al., PAKDD'07; Chui and Kao, PAKDD'08], we use dynamic programming (DP) to compute the probability that a source supports a sequence, and show that this suffices to compute the expected support of a sequential pattern. Next, we embed this DP algorithm into candidate generate-and-test approaches, and explore the pattern lattice both in a *breadth-first* (similar to GSP) and a *depth-first* (similar to SPAM) manner. We propose optimizations for efficiently computing the frequent 1-sequences, for re-using previously-computed results through *incremental support computation*, and for eliminating candidate sequences without computing their support via *probabilistic pruning*. Preliminary experiments show that our optimizations are effective in improving the CPU cost.

Key words: Mining Uncertain Data, Sequential Pattern Mining, Probabilistic Databases, Novel Algorithms for Mining, Theoretical Foundations of Data Mining.

1 Introduction

The problem of *sequential pattern mining (SPM)*, or finding frequent sequences of events in data with a temporal component, has been studied extensively [22, 16, 4] since its introduction in [17, 3]. In classical SPM, the data to be mined is deterministic, but it is recognized that data obtained from a wide range of data sources is inherently uncertain [1]. This paper is concerned with SPM in *probabilistic databases* [18], a popular framework for modelling uncertainty. Recently several data mining and ranking problems have been studied in this framework, including top-k [23, 8] and frequent itemset mining (FIM) [2, 5–7]. In classical SPM, the event database consists of tuples $\langle eid, e, \sigma \rangle$, where e is an *event*, σ is

* A shortened version of this report appeared in the proceedings of the 15th Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD 2011).

** On leave from Bahria University, Pakistan.

a *source* and *eid* is an *event-id* which incorporates a *time-stamp*. A tuple may record a retail transaction (event) by a customer (source), or an observation of an object/person (event) by a sensor/camera (source). Since event-ids have a time-stamp, the event database can be viewed as a collection of *source sequences*, one per source, containing a sequence of events (ordered by time-stamp) associated with that source, and classical SPM problem is to find patterns of events that have a temporal order that occur in a significant number of source sequences.

Uncertainty in SPM can occur in three different places: the source, the event and the time-stamp may all be uncertain (in contrast, in FIM, only the event can be uncertain). In a companion paper [15] the first two kinds of uncertainty in SPM were formalized as *source-level uncertainty (SLU)* and *event-level uncertainty (ELU)*, which we now summarize.

In SLU, the "source" attribute of each tuple is uncertain: each tuple contains a probability distribution over possible sources (*attribute-level* uncertainty [18]). As noted in [15] this formulation is applicable to scenarios such as the ambiguity arising when a customer makes a retail transaction, but the customer is not identified exactly, or because the customer database itself is probabilistic as a result of "deduplication" or cleaning [11]. In ELU, the source of the tuple is certain, but the events are uncertain. One example is the PEEX system [13] that aggregates unreliable observations of employees using RFID antennae at fixed locations into uncertain higher-level events such as "at time 103, Alice and Bob were having a meeting in room 435 with probability 0.4". Here, the source (Room 435) is fixed, but the event ({Alice, Bob}) is uncertain, and has only probability 0.4 of having occurred.

Furthermore, in [15] two measures of "frequentness", namely *expected support* and *probabilistic frequentness*, used for FIM in probabilistic databases [5, 7], were adapted to SPM, and the four possible combinations of models and measures were studied from a computational complexity viewpoint. This paper is focussed on efficient algorithms for the SPM problem in SLU probabilistic databases, under the expected support measure, and the contributions are as follows:

1. We give a dynamic-programming (DP) algorithm to determine efficiently the probability that a given source supports a sequence (*source support probability*), and show that this is enough to compute the expected support of a sequence in a SLU event database.
2. We give depth-first and breadth-first methods to find all frequent sequences in an SLU event database according to the expected support criterion.
3. To speed up the computation, we exploit properties of the DP to obtain algorithms for:
   (a) highly efficient computation of frequent 1-sequences,
   (b) *incremental computation* of the DP matrix, which allows us to minimize the amount of time spent on the expensive DP computation, and
   (c) *probabilistic pruning*, where we show how to rapidly compute an upper bound on the probability that a source supports a candidate sequence.
4. We empirically evaluate our algorithms, demonstrating their efficiency and scalability, as well as the effectiveness of the above optimizations.

Significance of Results. The source support probability algorithm ((1) above) shows that in probabilistic databases, FIM and SPM are very different – there is no need to use DP for FIM under the expected support measure [2, 6, 7].

Although the proof that source support probability allows the computation of the expected support of a sequence in an SLU database is simple, it is unexpected, since in SLU databases, there are dependencies between different sources – in any possible world, a given event can only belong to one source. In contrast, determining if a given sequence is *probabilistically* frequent in an SLU event database is #P-complete because of the dependencies between sources [15].

Finally, as noted in [15], (1) can be used to determine if a sequence is frequent in an ELU database using both frequentness criteria: expected support and probabilistic frequentness. This in turn implies efficient algorithms for enumerating frequent sequences under both frequentness criteria for ELU databases, and by using the framework of [10], we can also find maximal frequent sequential patterns in ELU databases.

The breadth-first and depth-first algorithms (2) have a high-level similarity to GSP [17] and SPADE/SPAM [22, 4]. However, since checking if a sequence is supported by a source requires a relatively expensive DP computation, significant modifications are needed to achieve good performance. Furthermore, it is unclear how to use either the projected database idea of PrefixSpan [16], or bitmaps as in SPAM. The efficiency of our algorithms depend instead upon the ideas ((3) above) of incremental computation, and probabilistic pruning. Although there is a high-level similarity between this pruning and a technique of [6] for FIM in probabilistic databases, the SPM problem is more complex, and our pruning rule is harder to obtain.

Related Work. Classical SPM has been studied extensively [17, 22, 16, 4]. Modelling uncertain data as *probabilistic* databases [18, 1] has led to several ranking/mining problems being studied in this context. The top-k problem (a ranking problem) has been studied intensively (see [12, 23, 8] and references therein). FIM in probabilistic databases was studied under the *expected* support measure in [2, 7, 6] and under the *probabilistic frequentness* measure in [5]. To the best of our knowledge, apart from [15], the SPM problem in probabilistic databases has not been studied. Uncertainty in the time-stamp attribute was considered in [19] – we do not consider time to be uncertain. Also [21] studies SPM in "noisy" sequences, but the model proposed there is very different to ours and does not fit in the probabilistic database framework.

2 Problem Statement

Classical SPM [17, 3]. Let $\mathcal{I} = \{i_1, i_2, \ldots, i_q\}$ be a set of *items* and $\mathcal{S} = \{1, \ldots, m\}$ be a set of *sources*. An *event* $e \subseteq \mathcal{I}$ is a collection of items. A *database* $D = \langle r_1, r_2, \ldots, r_n \rangle$ is an ordered list of *records* such that each $r_i \in D$ is of the form (eid_i, e_i, σ_i), where eid_i is a unique event-id, including a time-stamp (events are ordered by this time-stamp), e_i is an event and σ_i is a source.

A *sequence* $s = \langle s_1, s_2, \ldots, s_a \rangle$ is an ordered list of events. The events s_i in the sequence are called its *elements*. The *length* of a sequence s is the total number of items in it, i.e. $\sum_{j=1}^{a} |s_j|$; for any integer k, a k-sequence is a sequence of length k. Let $s = \langle s_1, s_2, \ldots, s_q \rangle$ and $t = \langle t_1, t_2, \ldots, t_r \rangle$ be two sequences. We say that s is a *subsequence* of t, denoted $s \preceq t$, if there exist integers $1 \leq i_1 < i_2 < \cdots < i_q \leq r$ such that $s_k \subseteq t_{i_j}$, for $k = 1, \ldots, q$. The *source sequence* corresponding to a source i is just the multiset $\{e|(eid, e, i) \in D\}$, ordered by *eid*. For a sequence s and source i, let $X_i(s, D)$ be an indicator variable, whose value is 1 if s is a subsequence of the source sequence for source i, and 0 otherwise. For any sequence s, define its *support* in D, denoted $Sup(s, D) = \sum_{i=1}^{m} X_i(s, D)$. The objective is to find all sequences s such that $Sup(s, D) \geq \theta m$ for some user-defined threshold $0 \leq \theta \leq 1$.

Probabilistic Databases. We define an SLU *probabilistic database* D^p to be an ordered list $\langle r_1, \ldots, r_n \rangle$ of records of the form (eid, e, W) where *eid* is an event-id, e is an event and W is a probability distribution over $\mathcal{S}$; the list is ordered by *eid*. The distribution W contains pairs of the form (σ, c), where $\sigma \in \mathcal{S}$ and $0 < c \leq 1$ is the confidence that the event e is associated with source σ and $\sum_{(\sigma,c)\in W} c = 1$. An example can be found in Table 1.

Table 1. A source-level uncertain database (L) transformed to p-sequences (R). Note that events like e_1 (marked with †) can only be associated with one of the sources X and Y in any possible world.

eid	e	W
e_1	(a, d)	$(X, 0.6)(Y, 0.4)$
e_2	(a)	$(Z : 1.0)$
e_3	(a, b)	$(X, 0.3)(Y, 0.2)(Z, 0.5)$
e_4	(b, c)	$(X, 0.7)(Z, 0.3)$

	p-sequence
D^p_X	$(a, d : 0.6)^{\dagger}(a, b : 0.3)(b, c : 0.7)$
D^p_Y	$(a, d : 0.4)^{\dagger}(a, b : 0.2)$
D^p_Z	$(a : 1.0)(a, b : 0.5)(b, c : 0.3)$

The *possible worlds* semantics of D^p is as follows. A *possible world* D^* of D^p is generated by taking each event e_i in turn, and assigning it to one of the possible sources $\sigma_i \in W_i$. Thus every record $r_i = (eid_i, e_i, W_i) \in D^p$ takes the form $r'_i = (eid_i, e_i, \sigma_i)$, for some $\sigma_i \in \mathcal{S}$ in D^*. By enumerating all such possible combinations, we get the complete set of possible worlds. We assume that the distributions associated with each record r_i in D^p are stochastically independent; the probability of a possible world D^* is therefore $\Pr[D^*] = \prod_{i=1}^{n} \Pr_{W_i}[\sigma_i]$. For example, a possible world D^* for the database of Table 1 can be generated by assigning events e_1, e_3 and e_4 to X with probabilities $0.6, 0.3$ and 0.7 respectively, and e_2 to Z with probability 1.0, and $\Pr[D^*] = 0.6 \times 1.0 \times 0.3 \times 0.7 = 0.126$. As every possible world is a (deterministic) database, concepts like the support of a sequence in a possible world are well-defined. The definition of the *expected*

support of a sequence s in D^p follows naturally:

$$ES(s, D^p) = \sum_{D^* \in PW(D^p)} \Pr[D^*] * Sup(s, D^*), \quad (1)$$

The problem we consider is:

> Given an SLU probabilistic database D^p, determine all sequences s such that $ES(s, D^p) \geq \theta m$, for some user-specified threshold θ, $0 \leq \theta \leq 1$.

Since there are potentially an exponential number of possible worlds, it is infeasible to compute $ES(s, D^p)$ directly using Eq. 1; next we show how to do this computation more efficiently using linearity of expectation and DP.

3 Computing Expected Support

p-sequences. A *p-sequence* is analogous to a source sequence in classical SPM, and is a sequence of the form $\langle (e_1, c_1) \dots (e_k, c_k) \rangle$, where e_j is an event and c_j is a confidence value. In examples, we write a p-sequence $\langle (\{a, d\}, 0.4), (\{a, b\}, 0.2) \rangle$ as $(a, d : 0.4)(a, b : 0.2)$. An SLU database D^p can be viewed as a collection of p-sequences $D_1^p, \dots, D_m^p$, where D_i^p is the p-sequence of source i, and contains a list of those events in D^p that have non-zero confidence of being assigned to source i, ordered by *eid*, together with the associated confidence (see Table 1(R)). However, the p-sequences corresponding to different sources are *not* independent, as illustrated in Table 1(R). Thus, one may view an SLU event database as a collection of p-sequences with dependencies in the form of x-tuples [8]. Nevertheless, we show that we can still process the p-sequences independently for the purpose of expected support computation:

$$\begin{aligned} ES(s, D^p) &= \textstyle\sum_{D^* \in PW(D^p)} \Pr[D^*] * Sup(s, D^*) = \sum_{D^*} \Pr[D^*] * \sum_{i=1}^{m} X_i(s, D^*) \\ &= \textstyle\sum_{i=1}^{m} \sum_{D^*} \Pr[D^*] * X_i(s, D^*) = \sum_{i=1}^{m} \mathrm{E}[X_i(s, D^p)], \end{aligned} \quad (2)$$

where E denotes the expected value of a random variable. Since X_i is a 0-1 variable, $\mathrm{E}[X_i(s, D^p)] = \Pr[s \preceq D_i^p]$, and we calculate the right-hand quantity, which we refer to as the *source support probability*. This cannot be done naively: e.g., if $D_i^p = (a, b : c_1)(a, b : c_2) \dots (a, b : c_q)$, then there are $O(q^{2k})$ ways in which a sequence $s = \langle \underbrace{(a)(a, b) \dots (a)(a, b)}_{k \text{ times}} \rangle$ could be supported by source i, and so we use DP.

Computing the Source Support Probability. Given a p-sequence $D_i^p = \langle (e_1, c_1), \dots, (e_r, c_r) \rangle$ and a sequence $s = \langle s_1, \dots, s_q \rangle$, we create a $(q+1) \times (r+1)$ matrix $A_{i,s}[0..q][0..r]$ (we omit the subscripts on A when the source and sequence are clear from the context). For $1 \leq k \leq q$ and $1 \leq \ell \leq r$, $A[k, \ell]$ will contain $\Pr[\langle s_1, \dots, s_k \rangle \preceq \langle (e_1, c_1), \dots, (e_\ell, c_\ell) \rangle]$. We set $A[0, \ell] = 1$ for all ℓ, $0 \leq \ell \leq r$

and $A[k,0] = 0$ for all $1 \leq k \leq q$, and compute the other values row-by-row. For $1 \leq k \leq q$ and $1 \leq \ell \leq r$, define:

$$c^*_{k\ell} = \begin{cases} c_\ell \text{ if } s_k \subseteq e_\ell \\ 0 \text{ otherwise} \end{cases} \tag{3}$$

The interpretation of Eq. 3 is that $c^*_{k\ell}$ is the probability that e_ℓ allows the element s_k to be matched in source i; this is 0 if $s_k \not\subseteq e_\ell$, and is otherwise equal to the probability that e_ℓ is associated with source i. Now we use the equation:

$$A[k,\ell] = (1 - c^*_{k\ell}) * A[k,\ell-1] + c^*_{k\ell} * A[k-1,\ell-1]. \tag{4}$$

Table 2 shows the computation of the source support probability of an example sequence $s = (a)(b)$ for source X in the probabilistic database of Table 1. Similarly, we can compute $\Pr[s \preceq D^p_Y] = 0.08$ and $\Pr[s \preceq D^p_Z] = 0.35$, so the expected support of $(a)(b)$ in the database of Table 1 is $0.558 + 0.08 + 0.35 = 1.288$.

Table 2. Computing $\Pr[s \preceq D^p_X]$ for $s = (a)(b)$ using DP in the database of Table 1.

	$(a, d : 0.6)$	$(a, b : 0.3)$	$(b, c : 0.7)$
(a)	$0.4 \times 0 + 0.6 \times 1 = 0.6$	$0.7 \times 0.6 + 0.3 \times 1 = \mathbf{0.72}$	0.72
$(a)(b)$	0	$0.7 \times 0 + 0.3 \times 0.6 = \mathit{0.18}$	$0.3 \times \mathit{0.18} + 0.7 \times \mathbf{0.72} = \mathbf{0.558}$

The reason Eq. 4 is correct is that if $s_k \not\subseteq e_\ell$ then the probability that $\langle s_1, \ldots, s_k \rangle \preceq \langle e_1, \ldots, e_\ell \rangle$ is the same as the probability that $\langle s_1, \ldots, s_k \rangle \preceq \langle e_1, \ldots, e_{\ell-1} \rangle$ (note that if $s_k \not\subseteq e_\ell$ then $c^*_{k\ell} = 0$ and $A[k,\ell] = A[k,\ell-1]$). Otherwise, $c^*_{k\ell} = c_\ell$, and we have to consider two *disjoint* sets of possible worlds: those where e_ℓ is not associated with source i (the first term in Eq. 4) and those where it is (the second term in Eq. 4). In summary:

Lemma 1. *Given a p-sequence D^p_i and a sequence s, by applying Eq. 4 repeatedly, we correctly compute* $\Pr[s \preceq D^p_i]$.

4 Optimizations

We now describe three optimized sub-routines for computing all frequent 1-sequences, for incremental support computation, and for probabilistic pruning.

Fast L_1 Computation. Given a 1-sequence $s = \langle (x) \rangle$, $x \in \mathcal{I}$, a simple closed-form expression for the probability with which source i supports s is:

$$\Pr[s \preceq D^p_i] = 1 - \prod_{\ell=1}^{r} (1 - c^*_{1\ell}) \ , \tag{5}$$

where $\Pr[s \preceq D_i^p]$ is the value $A[1, r]$. We now verify by induction that Eq. 4 gives the same answer as Eq. 5. As $A[1, \ell] = 1$ for all ℓ, $1 \leq \ell \leq r$, Eq. 4 can be written as:

$$A[1, \ell] = (1 - c_{1\ell}^*) * A[1, \ell - 1] + c_{1\ell}^*$$

We now prove Eq. 5 by induction, which clearly holds for $\ell = 1$, as $A[1, 1] = 1 - \prod_{\ell=1}^{1}(1 - c_{1\ell}^*) = 1 - (1 - c_{1\ell}^*) = c_{1\ell}^*$. Assume the induction hypothesis holds for $r = t - 1$, then:

$$\begin{aligned} A[1, t] &= (1 - c_{1t}^*) * A[1, t - 1] + c_{1t}^* \\ &= (1 - c_{1t}^*)(1 - \textstyle\prod_{\ell=1}^{t-1}(1 - c_{1\ell}^*)) + c_{1t}^* \quad (\textit{by induction hypothesis}) \\ &= 1 - \textstyle\prod_{\ell=1}^{t}(1 - c_{1\ell}^*) \ , \end{aligned}$$

which proves Eq. 5. This allows us to compute $ES(s, D^p)$ for *all* 1-sequences s in just one (linear-time) pass through D^p as follows. Initialize two arrays F and G, each of size $q = |\mathcal{I}|$, to zero and consider each source i in turn. If $D_i^p = \langle (e_1, c_1), \ldots, (e_r, c_r) \rangle$, for $k = 1, \ldots, r$ take the pair (e_k, c_k) and iterate through each $x \in e_k$, setting $F[x] := (F[x] * (1 - c_k)) + c_k$. Once we are finished with source i, if $F[x]$ is non-zero, we update $G[x] := G[x] + F[x]$ and reset $F[x]$ to zero (we use a linked list to keep track of which entries of F are non-zero for a given source). At the end, for any 1-sequence $s = \langle (x) \rangle$, where $x \in \mathcal{I}$, $G[x] = ES(\langle (x) \rangle, D^p)$.

Algorithm 1 Incremental Support Computation for I-extensions

```
1: B_{i,t}[0] = 0
2: for all ℓ = 1, ..., r do
3:     if t_q ⊈ e_ℓ then
4:         B_{i,t}[ℓ] = B_{i,t}[ℓ − 1]
5:     else
6:         B_{i,t}[ℓ] = (1 − c_ℓ) * B_{i,t}[ℓ − 1]+ (B_{i,s}[ℓ] − B_{i,s}[ℓ − 1] * (1 − c_ℓ))
```

Incremental Support Computation Let s and t be two sequences. Say that t is an *S-extension* of s if $t = s \cdot \{x\}$ for some item x, where $\cdot$ denotes concatenation (i.e. we obtain t by appending a single item as a new element to s). We say that t is an *I-extension* of s if $s = \langle s_1, \ldots, s_q \rangle$ and $t = \langle s_1, \ldots, s_q \cup \{x\} \rangle$ for some $x \notin s_q$, and x is lexicographically not less than any item in s_q (i.e. we obtain t by adding a new item to the last element of s). For example, if $s = (a)(b, c)$ and $x = d$, S- and I-extensions of s are $(a)(b, c)(d)$ and $(a)(b, c, d)$ respectively. Similar to classical SPM, we generate candidate sequences t that are either S- or I-extensions of existing frequent sequences s, and compute $ES(t, D^p)$ by computing $\Pr[t \preceq D_i^p]$ for all sources i. While computing $\Pr[t \preceq D_i^p]$, we will exploit the similarity between s and t to compute $\Pr[t \preceq D_i^p]$ more rapidly.

Let i be a source, $D_i^p = \langle (e_1, c_1), \ldots, (e_r, c_r) \rangle$, and $s = \langle s_1, \ldots, s_q \rangle$ be any sequence. Now let $A_{i,s}$ be the $(q+1) \times (r+1)$ DP matrix used to compute $\Pr[s \preceq D_i^p]$, and let $B_{i,s}$ denote the last row of $A_{i,s}$, that is, $B_{i,s}[\ell] = A_{i,s}[q, \ell]$ for $\ell = 0, \ldots, r$. We now show that if t is an extension of s, then we can quickly compute $B_{i,t}$ from $B_{i,s}$, and thereby obtain $\Pr[t \preceq D_i^p] = B_{i,t}[r]$:

Lemma 2. *Let s and t be sequences such that t is an extension of s, and let i be a source whose p-sequence has r elements in it. Then, given $B_{i,s}$ and D_i^p, we can compute $B_{i,t}$ in $O(r)$ time.*

Proof. If t is an s-extension of s, i.e. $t = s \cdot \{x\}$ for some item x, then $B_{i,s}$ is the last-but-one row of $A_{i,s}$, and we have the information needed to compute the last row (cf. Eq 4).

Now consider the case where t is a I-extension, i.e. $t = \langle s_1, \ldots, s_q \cup \{x\} \rangle$ for some $x \notin s_q$. Firstly, observe that since the first $q-1$ elements of s and t are pairwise equal, the first $q-1$ rows of $A_{i,s}$ and $A_{i,t}$ are also equal. The $(q-1)$-st row of $A_{i,s}$ is enough to compute the q-th row of $A_{i,t}$, but we only have $B_{i,s}$, the $q-1$-st row of $A_{i,s}$. In general we cannot calculate the entire $(q-1)$-st row of $A_{i,s}$ from the q-th row (i.e. we cannot "reverse" the DP calculation) but we can compute *enough* entries of $A_{i,s}$ to compute the q-th row of $A_{i,t}$.

We compute $A_{i,t}[q, \ell]$ for $\ell = 0, \ldots, r$ in that order. By convention, $A_{i,t}[q, 0] = 0$, so consider $\ell > 0$. If $t_q = s_q \cup \{x\} \not\subseteq e_\ell$, then $A_{i,t}[q, \ell] = A_{i,t}[q, \ell - 1]$, and we can move on to the next value of ℓ. If $t_q \subseteq e_\ell$, then $s_q \subseteq e_\ell$ and so:

$$A_{i,s}[q, \ell] = (1 - c_\ell) * A_{i,s}[q, \ell - 1] + c_\ell * A_{i,s}[q - 1, \ell - 1].$$

Since we know $B_{i,s}[\ell] = A_{i,s}[q, \ell]$, $B_{i,s}[\ell - 1] = A_{i,s}[q, \ell - 1]$ and c_ℓ, we can compute $A_{i,s}[q - 1, \ell - 1]$. But this value is equal to $A_{i,t}[q - 1, \ell - 1]$, which is the value from the $(q-1)$-st row of $A_{i,t}$ that we need to compute $A_{i,t}[q, \ell]$ or Specifically, $B_{i,t}[\ell]$. The pseudocode for this computation is given in Algorithm 1. An example is shown in Table 3.

Table 3. Example illustrating the incremental support computation of $B_{i,t}$ for $t = (a)(b, c)$ from $B_{i,s}$ where $s = (a)(b)$, by computing $\Pr[t \preceq D_X^p]$ in the database of Table 1. Note that the row corresponding to (a) is *not* available.

	$(a, d : 0.6)$	$(a, b : 0.3)$	$(b, c : 0.7)$
(a)	$0.4 \times 0 + 0.6 \times 1 = 0.6$	$0.7 \times 0.6 + 0.3 \times 1 = \mathbf{0.72}$	0.72
$(a)(b)$	0	$0.7 \times 0 + 0.3 \times 0.6 = 0.18$	$0.3 \times 0.18 + 0.7 \times \mathbf{0.72} = 0.558$
$(a)(b, c)$	0	0	$0.3 \times 0 + 0.7 \times \mathbf{0.72} = \mathbf{0.504}$

Probabilistic Pruning. We now describe a technique that allows us to prune non-frequent sequences s without fully computing $ES(s, D^p)$. For each source i, we obtain an upper bound on $\Pr[s \preceq D_i^p]$ and add up all the upper bounds; if the sum is below the threshold, s can be pruned. We first show:

Lemma 3. *Let $s = \langle s_1, \ldots, s_q \rangle$ be a sequence, and let D_i^p be a p-sequence. Then:*

$$\Pr[s \preceq D_i^p] \leq \Pr[\langle s_1, \ldots, s_{q-1} \rangle \preceq D_i^p] * \Pr[\langle s_q \rangle \preceq D_i^p].$$

Before proving Lemma 3, we indicate how it is used. Suppose, for example, that we have a candidate sequence $s = (a)(b,c)(a)$, and a source X. By Lemma 3:

$$\begin{aligned}\Pr[(a)(b,c)(a) \preceq D_X^p] &\leq \Pr[(a)(b,c) \preceq D_X^p] * \Pr[(a) \preceq D_X^p] \\ &\leq \Pr[(a) \preceq D_X^p] * \Pr[(b,c) \preceq D_X^p] * \Pr[(a) \preceq D_X^p] \\ &\leq (\Pr[(a) \preceq D_X^p])^2 * \min\{\Pr[(b) \preceq D_X^p], \Pr[(c) \preceq D_X^p]\}\end{aligned}$$

Observe that the quantities on the RHS are computed by the fast L_1 computation above, and can be stored in a small data structure associated with each source. Of course, if $\Pr[(a)(b,c) \preceq D_X^p]$ is available, an even tighter bound is $\Pr[(a)(b,c) \preceq D_X^p] * \Pr[(a) \preceq D_X^p]$. We now prove Lemma 3.

Proof. Let $A_{i,s}$ be the DP matrix for computing $\Pr[s \preceq D_i^p]$. For $\ell = 0, \ldots, r$, let $p_\ell = \Pr[s_q \preceq \langle (e_1, c_1), \ldots, (e_\ell, c_\ell) \rangle]$; p_r therefore is precisely $\Pr[\langle s_q \rangle \preceq D_i^p]$. As noted in Remark 1, we take $p_0 = 0$, and for $\ell = 1, \ldots, r$, we can compute p_ℓ using the equation: $p_\ell = (1 - c_{q\ell}^*)p_{\ell-1} + c_{q\ell}^*$, where $c_{q\ell}^*$ is as defined in Eq. 3. We now prove by induction on ℓ that:

$$A[q, \ell] \leq A[q-1, \ell] * p_\ell, \tag{6}$$

substituting $\ell = r$ in Eq. 6 proves the lemma. We now prove Eq. 6, which clearly holds for $\ell = 0$ since $A[q, 0] = A[q-1, 0] = p_0 = 0$. Subsequently:

$$\begin{aligned}A[q,\ell] &= (1 - c_{q\ell}^*) * A[q, \ell-1] + c_{q\ell}^* * A[q-1, \ell-1] \quad \text{(Eq. 4)} \\ &\leq (1 - c_{q\ell}^*) * A[q-1, \ell-1] * p_{\ell-1} + c_{q\ell}^* * A[q-1, \ell-1] \quad \text{(Eq. 6)} \\ &\leq A[q-1, \ell-1] * \left((1 - c_{q\ell}^*) * p_{\ell-1} + c_{q\ell}^*\right) \\ &\leq A[q-1, \ell-1] * p_\ell \leq A[q-1, \ell] * p_\ell.\end{aligned}$$

5 Candidate Generation

We now describe two candidate generation methods for enumerating all frequent sequences, one each based on breadth-first and depth-first exploration of the sequence lattice, which are similar to GSP [17, 3] and SPAM [4] respectively. We first note that an "Apriori" property holds in our setting:

Lemma 4. *Given two sequences s and t, and a probabilistic database D^p, if s is a subsequence of t, then $ES(s, D^p) \geq ES(t, D^p)$.*

Proof. In Eq. 1 note that for all $D^* \in PW(D^p)$, $Sup(s, D^*) \geq Sup(t, D^*)$.

Algorithm 2 Breadth-First Exploration

1: **Input:** SLU probabilistic database D^p and support threshold θ.
2: **Output:** All sequences s with $ES(s, D^p) \geq \theta m$.
3: $j \leftarrow 1$
4: $L_1 \leftarrow$ ComputeFrequent-1(D^p)
5: **while** $L_j \neq \emptyset$ **do**
6: $C_{j+1} \leftarrow$ Join L_j with itself
7: Prune C_{j+1}
8: **for all** $s \in C_{j+1}$ **do**
9: Compute $ES(s, D^p)$
10: $L_{j+1} \leftarrow$ all sequences $s \in C_{j+1}$ s.t. $ES(s, D^p) \geq \theta m\}$.
11: $j \leftarrow j + 1$
12: Stop and output $L_1 \cup \ldots \cup L_j$

5.1 Breadth-First Exploration

An overview of our BFS approach is in Algorithm 2. We now describe some details. Each execution of lines (8)-(12) is called a *phase*. Line 4 is done using the fast L_1 computation (see Section 4). Line 6 is done as in [17, 3]: two sequences s and s' in L_j are joined iff deleting the first item in s and the last item in s' results in the same sequence, and the result t comprises s extended with the last item in s'. This item is added as a new element if it was a separate element in s' (t is an S-extension of s) and is added to the last element of s otherwise (t is an I-extension). Thus, $(a)(b, c)(d)$ joins with $(b, c)(d, e)$ to yield $(a)(b, c)(d, e)$ and $(a, b)(c, d)$ joins with $(b)(c, d)(e)$ to yield $(a, b)(c, d)(e)$. The join of two 1-sequences (a) and (b), results in both $(a)(b)$ and (a, b) being output.

We apply apriori pruning to the set of candidates in the $(j+1)$-st phase, C_{j+1}, and probabilistic pruning can additionally be applied to C_{j+1} (note that apriori pruning has no effect on C_2, and probabilistic pruning is the only possibility).

Step 9 is divided into two main sub-steps. We consider each source i in turn and perform the following operations:

- Find a subset $N_{i,j+1}$ of C_{j+1} that may be supported by source i (this step is called *narrowing*). $N_{i,j+1}$ must include all sequences s in C_{j+1} such that $\Pr[s \preceq D_i^p]$ is strictly greater than zero.
- For all sequences in $s \in N_{i,j+1}$, compute $\Pr[s \preceq D_i^p]$ and update $ES(s, D^p)$.

Denote the set of all frequent j-sequences that have non-zero expected support in D_i^p by $L_{i,j}$. After computing L_1 in Step 1, we store $L_{i,1}$ for all i, for the entire duration of the algorithm, and with each $s \in L_{i,1}$, we also store $\Pr[s \preceq D_i^p]$ in case probabilistic pruning is used. We consider two ways of narrowing:

Prefix-Based Narrowing. We place all elements in C_{j+1} into a hash table, using its prefix of length j as the hash-key. Furthermore, we compute the sets $L_{i,j}$ for all sources i at the end of the j-th phase, and keep them until the end of the $(j+1)$-st phase. When considering source i, we iterate through all sequences $s \in L_{i,j}$,

and use the hash table to find all sequences t in C_{j+1} that are extensions of s. If t is created by adding the element x to s as an S-extension of s (i.e. $t = s \cdot \{x\}$) we further check to see if $x \in L_{i,1}$, and if so, we add t to $N_{i,j+1}$. If t is created as an I-extension of s, we add t to $N_{i,j+1}$ unconditionally.

Hashtree-Based Narrowing. We use a *hashtree* for the purpose of narrowing [17]. In any phase, a candidate sequence $s \in C_{j+1}$ is stored in the hashtree by hashing on each item in s in order, and the leaf node contains the $(j+1)$-st item and the candidate sequence s. When considering source i, we restrict our search to those parts of sequence-lattice that contain sequences which are likely to be supported by source i. We use $L_{i,1}$ for the purpose, and traverse hashtree recursively using $L_{i,1}$ only, until we have traversed all the leaf nodes, thus obtaining $N_{i,j+1}$.

Given $N_{i,j+1}$ we now discuss computing the support of t in source i. Since computing $\Pr[t \preceq D_i^p]$ is expensive and requires $j+1$ rows of a DP matrix to be computed, we attempt to reuse partial answers as follows. If we compute the support of t immediately after computing the support of s, where $s = \langle s_1, \ldots, s_q \rangle$ and $t = \langle t_1, \ldots, t_r \rangle$, then if s and t have a common prefix, i.e. for $k = 1, 2, \ldots, z$, $s_k = t_k$, then we start the computation of $\Pr[t \preceq D_i^p]$ from t_{z+1}. Our narrowing methods naturally tend to place sequences with common prefixes in consecutive positions of $N_{i,j+1}$.

Algorithm 3 Depth-First Exploration

Input: SLU probabilistic database D^p and support threshold θ.
Output: All sequences s with $ES(s, D^p) \geq \theta m$.
$L_1 \leftarrow$ ComputeFrequent-1(D^p)
for all sequences $x \in L_1$ **do**
    Call TraverseDFS(x)
Output all sequences in L

function TraverseDFS(s)
for all $x \in L_1$ **do**
    $t \leftarrow s \cdot \langle \{x\} \rangle$ {S-extension}
    Compute $ES(t, D^p)$
    if $ES(t, D^p) \geq \theta m$ **then**
        $L \leftarrow L \cup t$
        TraverseDFS(t)
    $t \leftarrow \langle s_1, \ldots, s_q \cup \{x\} \rangle$ {I-extension}
    Compute $ES(t, D^p)$
    if $ES(t, D^p) \geq \theta m$ **then**
        $L \leftarrow L \cup t$
        TraverseDFS(t)
end function

5.2 Depth-First Exploration

An overview of our depth-first approach is in Algorithm 3. We first compute the set of frequent 1-sequences, L_1 (line 3) and assume that L_1 is in ascending order. We then explore the pattern sub-lattice as shown, but use pruning to ensure that in line 8, unnecessary alternatives are not considered, as follows.

Consider a call of TraverseDFS(s), where s is some k-sequence. Prior to the support computation, we check all lexicographically smaller k-subsequences of t (which would have been explored previously) for frequentness, and reject t as infrequent if this test fails. We can then apply probabilistic pruning to t, and if t is still not pruned we compute its support (line 10). If at any stage t is found to be infrequent or if it is pruned, we do not consider x, the item used to extend s to t, as a possible alternative for S-extensions in the recursive tree under s (as in [4]). Observe that for sequences s and t, where t is an S- or I- extension of s, if $\Pr[s \preceq D_i^p] = 0$, then $\Pr[t \preceq D_i^p] = 0$. When computing $ES(s, D^p)$, we keep track of all the sources where $\Pr[s \preceq D_i^p] > 0$, denoted by $\mathcal{S}^s$. If s is frequent then when computing $ES(t, D^p)$, for any sequence t that is an S- or I- extension of s, we need only to visit the sources in $\mathcal{S}^s$.

Furthermore, with every source $i \in \mathcal{S}^s$, we assume that the array $B_{i,s}$ (see Section 4) has been saved prior to calling TraverseDFS(s), allowing us to use the incremental support computation. By implication, the arrays $B_{i,r}$ for all prefixes r of s are also stored for all sources $i \in \mathcal{S}^r$), so in the worst case, a source may store up to k arrays, if s is a k-sequence. The space usage of the DFS traversal is quite modest in practice, however.

Remark 1. The idea of focussing interest on only the sources $\mathcal{S}$ is of course quite similar to that of a *projected* database [16]. This, combined with the fast algorithm for computing frequent 1-sequences may suggest that candidate generation may be avoided altogether. However, note that in general one cannot simply perform a frequent 1-sequence computation on the projected database. For example, if an $\langle\{a\}\rangle$-projected database contained two p-sequences $(b : 0.5)(b : 0.5)(a : 0.5)(a : 0.5)$ and $(b : 0.5)(a : 0.5)(a : 0.5)(b : 0.5)$, then when considering whether $(a)(b)$ is frequent, it is not correct to compute the expected support of (b) in the projected database (for example, both p-sequences above would give the same contribution – 0.75 – to the support of (b) in the projected database, but clearly their support for $(a)(b)$ is different). Additional ideas are therefore needed to eliminate candidate generation altogether.

6 Experimental Evaluation

We report on an experimental evaluation of our algorithms in this section. Our implementations are in C# (Visual Studio .Net 2005), executed on a machine with a 3.2GHz Intel CPU and 3GB RAM running XP (SP3). Out reported running times are averages from multiple runs. We begin by describing datasets used for experiments. Then, we demonstrate the scalability of our algorithms, and also evaluate probabilistic pruning.

We use both real and synthetic datasets for our experiments. The real dataset *Gazelle* is from Blue Martini [14], and synthetic datasets are generated using the IBM Quest data generator [3]. We transform these deterministic datasets to probabilistic form using an approach similar to [2, 5, 23, 7]; we assign probabilities to each event in a source sequence using a uniform distribution over $(0, 1]$, thus obtaining a collection of p-sequences. Note that we in fact generate ELU data rather than SLU data: a key benefit of this approach to data generation is that it tends to preserve the distribution of frequent sequences in the deterministic data. On the other hand, the resource utilization of our algorithms is largely unaffected, as the algorithm for computing frequent sequences under the expected support measure is the same for ELU and SLU.

We follow the naming convention of [22]: a dataset named `C`i`D`j`K` means that the average number of events per source is i and the number of sources is j (in thousands). For example, the dataset `C10D20K` has on average 10 events per source and 20K sources. The alphabet size is fixed at 2K, and all other parameters, such as average items per event or length of the maximal sequential patterns have been set to default values.

We study three parameters in our experiments: the number of sources D, the average number of events per source C, and the threshold θ. We test our algorithms for one of the three parameters by keeping the other two fixed. Evidently, all other parameters being fixed, increasing the number of sources and the events per source, or decreasing θ, all make an instance harder. We choose the variants of our algorithms according to two "axes":

- Lattice traversal could be done using BFS with Prefix-based narrowing (pBFS) or BFS with Hashtree-based narrowing (hBFS), or DFS.
- Probabilistic Pruning (P) could be ON or OFF.

We thus report on six variants in all, for example "pBFS+P" represents the variant with BFS lattice traversal, with prefix-based narrowing and with probabilistic pruning ON.

Probabilistic Pruning. To show the effectiveness of probabilistic pruning, we kept statistics on the number of candidates in our algorithms for the datasets `C10D20K` and `C20D10K`. Table 4 shows that probabilistic pruning is highly effective at eliminating infrequent candidates in phase 2 — for example, in all variants, over 95% of infrequent candidates were eliminated without support computation.

In the subsequent phases, we compute upper bounds for probabilistic pruning in one of the two ways i.e. either by using $L_{i,1}$ probabilities only (in hBFS), or by using both $L_{i,1}$ and $L_{i,j}$ probabilities (in pBFS and DFS). As mentioned in Section 4 (after Lemma 3) that using both $L_{i,1}$ and $L_{i,j}$ probabilities gives a more accurate upper bound than only using $L_{i,1}$ probabilities, and our analysis in Table 4 supports the claim — for example, in `C10D20K` in Phase 3, $(223-91) = 132$ and $(234 - 91) = 143$ infrequent candidates that survived apriori pruning were reduced to $(167 - 91) = 76$ and $(175 - 91) = 84$, in pBFS and in DFS, respectively. The reduction was not as significant in hBFS as $(223 - 91) = 132$

infrequent candidates that survived apriori pruning were reduced to $(208-91) = 117$ only.

Similarly, in C20D10K in Phase 3, over 95% and over 35% of infrequent candidates that survived apriori pruning were eliminated by probabilistic pruning in DFS and in pBFS, respectively. In contrast, probabilistic pruning was less effective from Phase 3 onwards in hBFS compared to pBFS and DFS for the considered datasets and support threshold θ. We therefore, turn probabilistic pruning OFF after Phase 2 in hBFS in our experiments.

Table 4. Effectiveness of probabilistic pruning at $\theta = 2\%$, for datasets C10D20K (left) and C20D10K (right). On each side, the columns from left to right indicate the current phase, algorithm variant, numbers of candidates created by joining, remaining after apriori pruning, remaining after probabilistic pruning and deemed as frequent, respectively.

Dataset: C10D20K, θ=2%					
Ph.	Algo.	Joining	Apriori	Prob. prun.	Freq.
2	pBFS	15555	15555	173	39
2	hBFS	15555	15555	173	39
2	DFS	15555	15555	173	39
3	pBFS	237	223	167	91
3	hBFS	237	223	208	91
3	DFS	334	234	175	91
4	pBFS	381	337	261	44
4	hBFS	381	337	337	44
4	DFS	562	395	275	44

Dataset: C20D10K, θ=2%					
Ph.	Algo.	Joining	Apriori	Prob. prun.	Freq.
2	pBFS	283620	283620	2670	807
2	hBFS	283620	283620	2670	807
2	DFS	283620	283620	2670	807
3	pBFS	43257	4458	2985	394
3	hBFS	43257	4458	4410	394
3	DFS	122223	84126	3059	394
4	pBFS	2728	1802	1640	706
4	hBFS	2728	1802	1802	706
4	DFS	22480	4850	1726	706

In Fig. 1, we show the effect of probabilistic pruning on overall running time as θ decreases, for both synthetic (C10D10K) and real (Gazelle) datasets. It can be seen that pruning is effective particularly for low θ, both for synthetic and real datasets (note the speedup of DFS+P wrt DFS at $\theta \leq 1\%$) but gradually loses effectiveness for higher θ values.

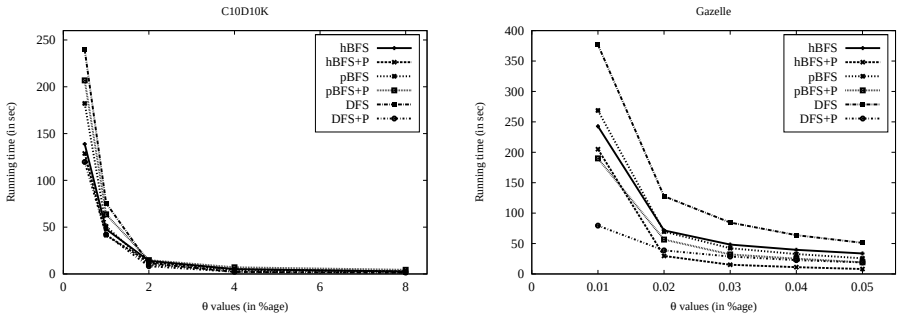

Fig. 1. Effectiveness of probabilistic pruning for decreasing values of θ, for the synthetic dataset (C10D10K) (left) and for real dataset Gazelle (right).

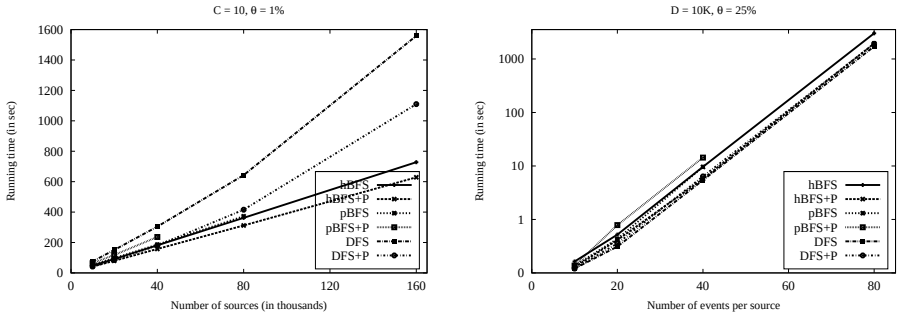

Fig. 2. Scalability for increasing number of sources (left), and for increasing number of events per source (right).

Scalability Testing. In Fig. 2(left), we fix the average number of events per source at 10 and $\theta = 1\%$, and test the scalability for increasing number of sources. We observe that all the algorithms scale essentially linearly with increasing number of sources. However, pBFS variants run out of memory for larger inputs (as indicated by missing dots) because of additional memory needed due to storing the sets $L_{i,j}$, which grow quite large. In Fig. 2(right), we test the scalability of our algorithms for increasing number of events per source by fixing the number of sources at 10K and $\theta = 25\%$. We observe behaviour similar to that reported in Fig. 2(right), as all variants scale well for increasing number of events per source; again, pBFS variants run out of memory for larger inputs. We do not see savings in CPU cost for DFS+P wrt DFS here, because θ is quite high (25%).

To conclude, we say that hBFS scales well overall for increasing number of sources, increasing number of events per source or decreasing values of θ. DFS also performs well and we see behaviour similar to hBFS. However, pBFS has high memory requirements and runs out of memory for larger inputs.

7 Conclusions and Future Work

We have considered the problem of finding all frequent sequences in SLU probabilistic databases. This is a first study on efficient algorithms for this problem, and naturally a number of open directions remain, including expanding the range of "interesting objects", e.g. finding maximal frequent sequences or having a more restricted definition of the "subsequence" relation. However, equally challenging is exploring further the notion of "interestingness". In this paper, we have used the expected support measure, which has the advantage that it can be computed efficiently for SLU databases – the probabilistic frequentness [5] is provably intractable for SLU databases [15]. Our approach yields (in principle) efficient algorithms for both measures in ELU databases, and comparing both measures in terms of computational cost versus solution quality is an interesting future direction. A number of other longer-term challenges remain, including

creating a data generator that gives an "interesting" SLU database and considering more general models of uncertainty (e.g. it is not clear that the assumption of independence between successive uncertain events is justified).

References

1. Aggarwal, C.C. (ed.): Managing and Mining Uncertain Data. Springer (2009)
2. Aggarwal, C.C., Li, Y., Wang, J., Wang, J.: Frequent pattern mining with uncertain data. In: Elder et al. [9], pp. 29–38
3. Agrawal, R., Srikant, R.: Mining sequential patterns. In: Yu, P.S., Chen, A.L.P. (eds.) ICDE. pp. 3–14. IEEE Computer Society (1995)
4. Ayres, J., Flannick, J., Gehrke, J., Yiu, T.: Sequential pattern mining using a bitmap representation. In: KDD. pp. 429–435 (2002)
5. Bernecker, T., Kriegel, H.P., Renz, M., Verhein, F., Züfle, A.: Probabilistic frequent itemset mining in uncertain databases. In: Elder et al. [9], pp. 119–128
6. Chui, C.K., Kao, B.: A decremental approach for mining frequent itemsets from uncertain data. In: PAKDD. pp. 64–75 (2008)
7. Chui, C.K., Kao, B., Hung, E.: Mining frequent itemsets from uncertain data. In: Zhou, Z.H., Li, H., Yang, Q. (eds.) PAKDD. LNCS, vol. 4426, pp. 47–58. Springer (2007)
8. Cormode, G., Li, F., Yi, K.: Semantics of ranking queries for probabilistic data and expected ranks. In: ICDE. pp. 305–316. IEEE (2009)
9. Elder, J.F., Fogelman-Soulié, F., Flach, P.A., Zaki, M.J. (eds.): Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Paris, France, June 28 - July 1, 2009. ACM (2009)
10. Gunopulos, D., Khardon, R., Mannila, H., Saluja, S., Toivonen, H., Sharm, R.S.: Discovering all most specific sentences. ACM Trans. DB Syst. 28(2), 140–174 (2003)
11. Hassanzadeh, O., Miller, R.J.: Creating probabilistic databases from duplicated data. The VLDB Journal 18(5), 1141–1166 (2009)
12. Hua, M., Pei, J., Zhang, W., Lin, X.: Ranking queries on uncertain data: a probabilistic threshold approach. In: Wang [20], pp. 673–686
13. Khoussainova, N., Balazinska, M., Suciu, D.: Probabilistic event extraction from RFID data. In: ICDE. pp. 1480–1482. IEEE (2008)
14. Kohavi, R., Brodley, C., Frasca, B., Mason, L., Zheng, Z.: KDD-Cup 2000 organizers' report: Peeling the onion. SIGKDD Explorations 2(2), 86–98 (2000)
15. Muzammal, M., Raman, R.: On probabilistic models for uncertain sequential pattern mining. In: Cao, L., Feng, Y., Zhong, J. (eds.) ADMA (1). LNCS, vol. 6440, pp. 60–72. Springer (2010)
16. Pei, J., Han, J., Mortazavi-Asl, B., Wang, J., Pinto, H., Chen, Q., Dayal, U., Hsu, M.: Mining sequential patterns by pattern-growth: The PrefixSpan approach. IEEE Trans. Knowl. Data Eng. 16(11), 1424–1440 (2004)
17. Srikant, R., Agrawal, R.: Mining sequential patterns: Generalizations and performance improvements. In: Apers, P.M.G., Bouzeghoub, M., Gardarin, G. (eds.) EDBT. LNCS, vol. 1057, pp. 3–17. Springer (1996)
18. Suciu, D., Dalvi, N.N.: Foundations of probabilistic answers to queries. In: Özcan, F. (ed.) SIGMOD Conference. p. 963. ACM (2005)
19. Sun, X., Orlowska, M.E., Li, X.: Introducing uncertainty into pattern discovery in temporal event sequences. In: ICDM. pp. 299–306. IEEE Computer Society (2003)

20. Wang, J.T.L. (ed.): Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2008, Vancouver, BC, Canada, June 10-12, 2008. ACM (2008)
21. Yang, J., Wang, W., Yu, P.S., Han, J.: Mining long sequential patterns in a noisy environment. In: Franklin, M.J., Moon, B., Ailamaki, A. (eds.) SIGMOD Conference. pp. 406–417. ACM (2002)
22. Zaki, M.J.: SPADE: An efficient algorithm for mining frequent sequences. Machine Learning 42(1/2), 31–60 (2001)
23. Zhang, Q., Li, F., Yi, K.: Finding frequent items in probabilistic data. In: Wang [20], pp. 819–832