Mining Sequential Patterns from Probabilistic Databases by Pattern-Growth

Muhammad Muzammal

Department of Computer Science, University of Leicester, UK.
mm386@mcs.le.ac.uk

Abstract. We propose a *pattern-growth* approach for mining *sequential patterns* from *probabilistic databases*. Our considered model of uncertainty is about the situations where there is uncertainty in associating an *event* with a *source*; and consider the problem of enumerating all sequences whose *expected support* satisfies a user-defined threshold θ. In an earlier work [Muzammal and Raman, PAKDD'11], adapted representative candidate generate-and-test approaches, GSP (breadth-first sequence lattice traversal) and SPADE/SPAM (depth-first sequence lattice traversal) to the probabilistic case. The authors also noted the difficulties in generalizing PrefixSpan to the probabilistic case (PrefixSpan is a pattern-growth algorithm, considered to be the best performer for deterministic sequential pattern mining). We overcome these difficulties in this note and adapt PrefixSpan to work under probabilistic settings. We then report on an experimental evaluation of the candidate generate-and-test approaches against the pattern-growth approach.

Key words: Mining Uncertain Data, Mining complex sequential data, Probabilistic Databases, Novel models and algorithms.

1 Introduction

Agrawal and Srikant [14, 2] defined the problem of *Sequential Pattern Mining (SPM)*, which involves discovery of frequent sequences of events in data with a temporal component; SPM has become a classical and well-studied problem in data mining [17, 13, 3]. In classical SPM, the database to be mined consists of tuples $\langle eid, e, \sigma \rangle$, where e is an *event*, σ is 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). All of the components of the tuple are assumed to be *certain*, or completely determined.

However, 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* [15], a popular framework for modelling uncertainty. Recently several data mining and ranking problems have been studied in this framework, including top-k [18, 5], frequent itemset mining (FIM) [1, 4] and sequential pattern mining [11, 12]. In [11] two kinds of uncertainty in SPM were formalized:

source-level uncertainty (SLU) and *event-level uncertainty (ELU)*. In SLU, the "source" attribute of each tuple is uncertain: each tuple contains a probability distribution over possible sources (*attribute-level* uncertainty [15]). As noted in [11], this formulation applies to scenarios where it is known that some customer made a specific retail transaction, but the identity of the customer who made that transaction is uncertain. This could happen because of incomplete customer details, or because the customer database itself is probabilistic as a result of "deduplication" or cleaning [7]. In ELU, the source of the tuple is certain, but the events are uncertain. This applies to several scenarios involving sensors in fixed locations detecting events using noisy techniques. In this case, either the existence of an event is uncertain (for example tracking endangered animals using sensors [8] or aggregating unreliable observations into events [9]) or the event's existence is essentially certain, but its content is uncertain (for example, reading a numberplate using automated numberplate recognition or sequences of search terms that may have ambiguous meanings [11]).

Our contributions. In [12], efficient algorithms were proposed for the SPM problem in SLU probabilistic databases, under the *expected support* measure. These algorithms were based on the *candidate generate-and-test* approach, which is known to be relatively inefficient for classical SPM. In [12], it was noted that it is not straightforward to adapt the *pattern-growth* approach [13], which is usually the best for classical SPM, to the probabilistic case. In this paper, we overcome the obstacles mentioned in [12], and propose a pattern-growth method for the SPM problem in probabilistic databases. In classical SPM, pattern-growth works by performing L_1 *computation* on a *projected* database. The key contributions of this work are to formulate the analogue of a projected database in probabilistic settings, and to identify the appropriate L_1 computation to perform on the projected database. Unlike the deterministic case, it appears that pattern-growth for probabilistic SPM appears to require additional memory. We have implemented the new pattern-growth algorithm and present an experimental evaluation of the pattern-growth algorithm against the ones presented in [12]; this evaluation shows that although pattern-growth is superior to candidate generation in the deterministic settings, the picture is not as clear as for the probabilistic case.

2 Problem Statement

Classical SPM [14, 2]. 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* D_i corresponding to a source i is just the multiset $\{e|(eid, e, i) \in D\}$, ordered by *eid*. For any sequence s, define its *support* in D, denoted $Sup(s, D)$ is the number of sources i such that $s \preceq D_i$. The objective is to find all sequences s such that $Sup(s, D) \geq \theta m$ for some user-defined threshold $0 < \theta \leq 1$.

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(L). The *possible worlds* semantics of D^p

Table 1. An SLU event database (L) transformed to p-sequences (R). Note that the events like e_1 (marked with † on (R)) can only be associated with one of the sources X, Y and Z in any possible world.

eid	event	W
e_1	(a, c, e)	$(X : 0.1)(Y : 0.6)(Z : 0.3)$
e_2	(b, c, d)	$(X : 0.7)(Y : 0.3)$
e_3	(a, d, e)	$(X : 0.2)(Y : 0.3)(Z : 0.5)$
e_4	(b, c, e)	$(X : 0.6)(Z : 0.4)$

	p-sequence
D^p_X	$(a, c, e : 0.1)^\dagger (b, c, d : 0.7)(a, d, e : 0.2)$ $(b, c, e : 0.6)$
D^p_Y	$(a, c, e : 0.6)^\dagger (b, c, d : 0.3)(a, d, e : 0.3)$
D^p_Z	$(a, c, e : 0.3)^\dagger (a, d, e : 0.5)(b, c, e : 0.4)$

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^*. The complete set of possible worlds is obtained by enumerating all such possible combinations. We assume that the distributions W_i 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 event e_1 to Z with probability 0.3, events e_2 and e_4 to X with probabilities 0.7 and 0.6 respectively, and event e_3 to Y with probability 0.3, and $\Pr[D^*] = 0.3 \times 0.7 \times 0.3 \times 0.6 = 0.0378$.

As a possible world is a deterministic instance of a given probabilistic database, concepts like the support of a sequence in a possible world do apply. The *expected support* of a sequence s in D^p is computed as follows:

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

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, $0 < \theta \leq 1$.

Observe that it is not feasible to use Eq. 1 directly due to the exponential number of possible worlds. Consider for example, we want to compute the probability with which source X supports a sequence $s = (a)(b)$ in the sample database of Table 1. There can be two ways in which s can be supported by source X i.e. either e_1 does support (a) and at least one of the events e_2 or e_4 support (b) or alternatively, e_1 does not support (a) but e_3 does support (a), and e_4 supports (b). The probability that X supports s is calculated as 0.196. Clearly, computing the source support probability this way is an expensive operation. In [12], a dynamic programming (DP) based algorithm was proposed to compute the source support probability $\Pr[s \preceq D_i^p]$, and it was shown that the ES of a sequence could be computed as follows:

$$ES(s, D^p) = \sum_{i=1}^{m} \Pr[s \preceq D_i^p] \tag{2}$$

3 Pattern-Growth Approach

We now describe our Pattern-Growth-Approach (PGA) that uses in essence the already proposed sub-routines, Dynamic Programming (DP) and fast L_1 computation [12]; and integrates it with the pattern-growth mechanism [13]. We first review some concepts:

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. 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 associated with source i, ordered by *eid*, together with the associated confidence (see Table 1(R)). Further, given a sequence $s = \langle s_1, \dots, s_q \rangle$ and an item x, s can either be extended by adding x as a separate element in s, $s \cdot \{x\}$ (S-extension) or by appending x to the last element in s, $\langle s_1, \dots, s_q \cup \{x\} \rangle$ (I-extension). For example, for $s = (a)(b)$ and $x = c$, S- and I-extensions of s are $(a)(b)(c)$ and $(a)(b, c)$ respectively.

Dynamic Programming. Let i be a source, $D_i^p = \langle (e_1, c_1), \dots, (e_r, c_r) \rangle$, and $s = \langle s_1, \dots, s_q \rangle$ be any sequence. Now let $A_{i,s}$ be the $(q \times r)$ 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 $\ell = 1, \dots, r$. For $1 \leq k \leq q$ and $1 \leq \ell \leq r$, $A[k, \ell]$ will contain $\Pr[\langle s_1, \dots, s_k \rangle \preceq \langle (e_1, c_1), \dots, (e_\ell, c_\ell) \rangle]$, so $A[q, r]$ is the value $\Pr[s \preceq D_i^p]$. We set $A[1, \ell] = 1$ for all ℓ, $1 \leq \ell \leq r$ and $A[k, 1] = 0$ 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:

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

where $c^*_{k\ell}$ is the probability that the element s_k is contained in the event e_ℓ in source i; If $s_k \subseteq e_\ell$, $c^*_{k\ell}$ is equal to the probability that e_ℓ is associated with source i, and 0 otherwise. Next, use the following recurrence:

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

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

fast L_1 computation. It was shown by [12] that it was possible to compute all frequent 1-sequences in a single pass over the database. The procedure for this is as follows: Initialize two arrays F and G, each of size $q = |\mathcal{I}|$, to zero and consider each source i in turn. If $D^p_i = \langle (e_1, c_1), \ldots, (e_r, c_r)\rangle$, 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 finished with source i, if $F[x]$ is non-zero, update $G[x] := G[x] + F[x]$ and reset $F[x]$ to zero (for each source i use a list structure to keep track of all the non-zero entries in F). Finally, for any item $x \in \mathcal{I}$, $G[x] = ES(\langle x\rangle, D^p)$.

PrefixSpan. PrefixSpan is based on the idea of pattern-growth, and works as follows: First, all frequent 1-sequences are discovered. It is argued that any of the frequent 2-sequences must begin with a frequent 1-sequence and therefore, the complete set of sequential patterns can be partitioned into as many subsets as the number of frequent 1-sequences where each 1-sequence is taken as a prefix. A *projected database* is a *smaller* databased based on some prefix (sequence). For example, in the sample database of Table 1(R), a (d)-projected database is $\{\langle (a, d, e : 0.2)(b, c, e : 0.6)\rangle, \langle (a, d, e : 0.3)\rangle, \langle (\_\,, e : 0.5)(b, c, e : 0.4)\rangle\}$. The subset of sequential patterns is mined by constructing the set of projected databases based on frequent 1-sequences and mining each recursively. For example, if (e) is a frequent 1-sequence in the above (d)-projected database, a $(d)(e)$-projected database looks like $\{\langle (b, c, e : 0.6)\rangle, \langle\rangle, \langle\rangle\}$. This recursive mining process continues until no more sequential patterns could be found. For details see [13].

It was noted in [12] that it is not correct to simply perform the fast L_1 computation on a projected database. For example, if an (a)-projected database contained two p-sequences $(b : 0.5)(b : 0.5)(a : 0.5)$ and $(b : 0.5)(a : 0.5)(b : 0.5)$, 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). In this work, we show how these sub-routines could be put together to find all frequent sequential patterns using PGA.

3.1 Pattern-Growth Step

Pre-conditions:

1. s is a previously discovered frequent sequence.

Table 2. An example of computing the ES of all S-extensions of $s = (a)$ for source X in the sample database of Table 1. The gray row is the $B_{X,s}$ array. In the bottom half, the cells that changed in F' after processing the corresponding event are marked as gray. After processing source X, values in the final column are updated to G'.

D^p_X	$(a, c, e : 0.1)$	$(b, c, d : 0.7)$	$(a, d, e : 0.2)$	$(b, c, e : 0.6)$
(a)	0.1	0.1	0.28	0.28
(a)	0.0	0.0	0.02	0.020
(b)	0.0	0.07	0.07	0.196
(c)	0.0	0.07	0.07	0.196
(d)	0.0	0.07	0.076	0.076
(e)	0.0	0.0	0.02	0.176
	(i)	(ii)	(iii)	(iv)

2. The list of sources i, where $\Pr[s \preceq D^p_i] > 0$ is available.
3. The $B_{i,s}$ arrays for all such sources i are also available.

Objective: To compute the ES of all the S- or I-extensions of s in one pass over the database, and thus discover all frequent extensions of s.

Steps: We consider the two cases of finding the S- and I-extensions of s in turn. We compute the frequent S-extensions of s as follows: Let i be a source, $D^p_i = \langle (e_1, c_1), \ldots, (e_r, c_r) \rangle$, and $s = \langle s_1, \ldots, s_q \rangle$ be any sequence. Initialize two arrays F' and G', each of size $q = |\mathcal{I}|$ to zero and consider each source i in turn. Then scan $B_{i,s}$ up-to the first non-zero entry e_k, and for every item x in e_ℓ, $k < \ell \leq r$, update $F'[x]$ as follows:

$$F'[x] := ((1 - c_\ell) * F'[x]) + (c_\ell * B_{i,s}[\ell - 1]) \tag{5}$$

We keep track of all the non-zero entries in $F'[x]$, and once finished with source i, update $G'[x] := G'[x] + F'[x]$ and reset $F'[x]$ to zero. After all the sources i are processed, all the entries in $G'[x] \geq \theta m$ are frequent S-extensions of s. An example of this computation is shown in Table 2.

For the I-extensions case, the initializations are the same as for the S-extensions case. For a sequence $s = \langle s_1, \ldots, s_q \rangle$ and source i, when scanning $B_{i,s}$ up-to the first non-zero entry e_ℓ which means $s_q \subseteq e_\ell$, for every item x in e_ℓ that is not in s_q and is lexicographically greater than all items in s_q, update $F'[x]$ as follows:

$$F'[x] := (1 - c_\ell) * F'[x] + (B_{i,s}[\ell] - B_{i,s}[\ell - 1] * (1 - c_\ell)), \tag{6}$$

and apply Eq. 6 to all the events e_ℓ, $\ell \leq r$, where $s_q \subseteq e_\ell$ (or alternatively where the $B_{i,s}$ values change). After all the sources i are processed, all the entries in $G'[x] \geq \theta m$ are frequent I-extensions of s.

Algorithm 1 Pattern-Growth Approach

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: $L_1 \leftarrow$ ComputeFrequent-1-sequences(D^p)
4: **for all** sequences $x \in L_1$ **do**
5: Compute $B_{i,x}$ arrays
6: Call ProjectedDB(x)
7: **function** ProjectedDB(s)
8: $L_S \leftarrow$ Compute Frequent S-extensions {fast L_1 computation}
9: $L_I \leftarrow$ Compute Frequent I-extensions {fast L_1 computation}
10: Output all Frequent Sequences {s extended with x, for all x in L_S and L_I}
11: **for all** $x \in L_S$ **do**
12: $t \leftarrow \langle s \cdot \{x\} \rangle$ {S-extension}
13: Compute $B_{i,t}$ arrays
14: ProjectedDB(t)
15: **for all** $x \in L_I$ **do**
16: $t \leftarrow \langle s_1, \ldots, s_q \cup \{x\} \rangle$ {I-extension}
17: Compute $B_{i,t}$ arrays
18: ProjectedDB(t)
19: **end function**

Pattern-Growth Algorithm. An overview of our pattern-growth algorithm is in Fig. 1. We first compute the set of frequent 1-sequences, L_1 (Line 3) (assume L_1 is in ascending order). For each 1-sequence x, first we compute the $B_{i,x}$ arrays for each source (Line 5) and also keep track of all the sources where $\Pr[x \preceq D_i^p] > 0$ (projected database) and then, call the ProjectedDB(x) sub-routine (Line 6).

In the ProjectedDB sub-routine, we first compute all the frequent S- and I-extensions of s using the fast L_1 computation by applying Eq. 5 and Eq. 6 accordingly (Line 8 and 9). In step 10, output all the frequent S- and I-extensions of s computed in the previous steps. In steps 11-18, for every sequence t which is a frequent S- or I-extension of s, compute $B_{i,t}$ arrays and also keep track of all the sources where $\Pr[t \preceq D_i^p] > 0$, and call the ProjectedDB sub-routine recursively to mine all frequent sequential patterns.

Table 3. Number of DP computations (in millions) performed by each algorithm, for the set of experiments in Fig. 1 (L) and in Fig. 2 (R).

C10D10K	BFS+P	DFS+P	PGA
$\theta = 0.5$ %	3.141	3.199	1.494
$\theta = 1$ %	1.711	1.465	0.886
$\theta = 2$ %	0.879	0.781	0.487
$\theta = 4$ %	0.505	0.487	0.281
gazelle			
$\theta = 0.01$ %	0.777	0.375	0.261
$\theta = 0.02$ %	0.149	0.172	0.152
$\theta = 0.03$ %	0.073	0.129	0.122
$\theta = 0.04$ %	0.045	0.110	0.107

$C = 10, \theta = 1\%$	BFS+P	DFS+P	PGA
$D = 10$K	1.465	1.711	0.886
$D = 20$K	2.890	3.370	1.735
$D = 40$K	5.716	6.718	3.464
$D = 80$K	11.460	13.423	6.936
$D = 10K, \theta = 25\%$			
$C = 10$	0.100	0.111	0.081
$C = 20$	0.690	0.694	0.314
$C = 40$	13.044	13.891	4.868
$C = 80$	2353.677	2782.807	881.480

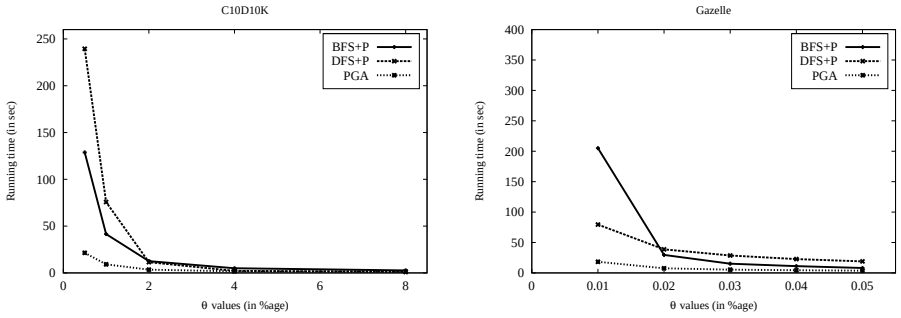

Fig. 1. Scalability of the three algorithms for decreasing values of θ, for synthetic dataset (`C10D10K`) (L) and for real dataset Gazelle (R).

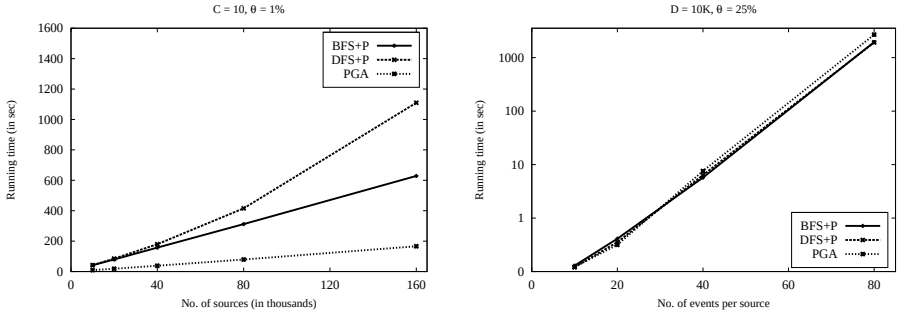

Fig. 2. Scalability for increasing number of sources D, with average number of events per source $C = 10$ and ES $= 1\%$ (L), and for increasing number of events per sources C with number of sources $D = 10K$ and ES $= 25\%$ (R).

4 Experimental Evaluation

In [12], the authors adapt GSP and SPADE/SPAM to yield a breadth-first (BFS) and a depth-first (DFS) algorithm, respectively. In addition, they also propose a probabilistic pruning technique to eliminate potential infrequent candidates without support computation, achieving an overall speedup. We therefore, choose two of the faster candidate generation variants from [12] i.e. BFS+P (breadth-first search with pruning) and DFS+P (depth-first search with pruning), and compare these with the Pattern-Growth Approach (PGA) proposed in this work.

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). We begin by describing the datasets used for experiments. Then, we demonstrate the scalability of the three algorithms. Our reported running times are averages from multiple runs. In our experiments, we use both real (*gazelle* from Blue Martini [10]) and synthetic (IBM Quest [2]) datasets. We transform these deterministic datasets to probabilistic form in a way similar to [1, 4, 18, 12]; we assign prob-

abilities to each event in a source sequence using a uniform distribution over $(0, 1]$, thus obtaining a collection of p-sequences.

The real dataset Gazelle has 29369 sequences and 35722 events. For synthetic datasets, we follow the naming convention of [17]: a dataset named CiDjK means that the average number of events per source is i and the number of sources is j (in thousands). Alphabet size is 2K and all other parameters are set to default. For example, the dataset C10D20K has on average 10 events per source and 20K sources. We consider following three parameters in our experiments: number of sources D, average number of events per source C, and support threshold θ. We test our algorithms for increasing D, increasing C, and decreasing θ values by keeping the other two parameters fixed.

Scalability Testing. In the first set of experiments, we fix $C = 10$ and $\theta = 1\%$, and test the scalability of these algorithms for decreasing θ values. We report our results for synthetic dataset (C10D10K) in Fig. 1(L), and for real dataset (Gazelle) in Fig. 1(R). It can be seen that PGA performs better than both the candidate generate-and-test approaches for real as well as for synthetic dataset. The performace difference is more obvious for harder instances (at low θ values).

In another set of experiments, we test the scalability of these algorithms for increasing values of D by fixing $C = 10$ and $\theta = 1\%$ (Fig. 2(L)), and by fixing $D = 10K$ and $\theta = 25\%$, for increasing values of C (Fig. 2(R)). It can be seen that PGA performs better than the other two algorithms for increasing D (Fig. 2(L)). However, we do not see improvements for increasing C (Fig. 2(R)). We are currently investigating the reasons for this behaviour. Note that DFS+P processes only one S- or I-extension of s at a time, whereas in PGA all the extensions of s are processed simultaneously.

We also kept statistics about the number of DP computations for each algorithm (Table. 3). The datasets and support thresholds are the same as in Fig. 1 and Fig. 2. We observe that PGA performs the least number of DP computations consistently, as in PGA the $B_{i,s}$ arrays are computed only for the frequent sequences. As noted in [13] that candidate generation approaches suffer from an exponential number of candidates at low θ values, this cost is even higher in probabilistic case because of the $B_{i,s}$ arrays. Further, note that there is no need for keeping $B_{i,s}$ arrays for BFS in contrast with the PGA and therefore, PGA has additional memory needs.

5 Conclusions and Future Work

We have considered the problem of finding all frequent sequences by pattern-growth in SLU databases. We have evaluated PGA in contrast with the candidate generate-and-test approaches, and we observe that the PGA performs better than the candidate generation approaches in general. The speedup in running time can be seen for the real dataset, in particular at low θ values, and for the synthetic datasets as well when the source sequences are not very long or for low θ values. The statistics about the number of DP computations also show that the

PGA performs the least number of DP computations consistently. We conclude that PGA is generally efficient than the candidate generation algorithms. In future, we intend to investigate that why the performance difference is not so obvious when the source sequences are very long or for higher θ values, and whether PGA has similar behaviour for other real datasets.

References

1. Aggarwal, C.C. (ed.): Managing and Mining Uncertain Data. Springer (2009)
2. Agrawal, R., Srikant, R.: Mining sequential patterns. In: Yu, P.S., Chen, A.L.P. (eds.) ICDE. pp. 3–14. IEEE Computer Society (1995)
3. Ayres, J., Flannick, J., Gehrke, J., Yiu, T.: Sequential pattern mining using a bitmap representation. In: KDD. pp. 429–435. ACM (2002)
4. Bernecker, T., Kriegel, H.P., Renz, M., Verhein, F., Züfle, A.: Probabilistic frequent itemset mining in uncertain databases. In: Elder et al. [6], pp. 119–128
5. Cormode, G., Li, F., Yi, K.: Semantics of ranking queries for probabilistic data and expected ranks. In: ICDE. pp. 305–316. IEEE (2009)
6. 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)
7. Hassanzadeh, O., Miller, R.J.: Creating probabilistic databases from duplicated data. The VLDB Journal 18(5), 1141–1166 (2009)
8. Hua, M., Pei, J., Zhang, W., Lin, X.: Ranking queries on uncertain data: a probabilistic threshold approach. In: Wang [16], pp. 673–686
9. Khoussainova, N., Balazinska, M., Suciu, D.: Probabilistic event extraction from RFID data. In: ICDE. pp. 1480–1482. IEEE (2008)
10. 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)
11. 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)
12. Muzammal, M., Raman, R.: Mining sequential patterns from probabilistic databases. In: Huang, J.Z., Cao, L., Srivastava, J. (eds.) PAKDD (II). LNAI, vol. 6635, pp. 210–221. Springer (2011)
13. 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)
14. 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)
15. Suciu, D., Dalvi, N.N.: Foundations of probabilistic answers to queries. In: Özcan, F. (ed.) SIGMOD Conference. p. 963. ACM (2005)
16. 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)
17. Zaki, M.J.: SPADE: An efficient algorithm for mining frequent sequences. Machine Learning 42(1/2), 31–60 (2001)
18. Zhang, Q., Li, F., Yi, K.: Finding frequent items in probabilistic data. In: Wang [16], pp. 819–832