Knowl Inf Syst (2015) 44:325–358
DOI 10.1007/s10115-014-0766-7

REGULAR PAPER

Mining sequential patterns from probabilistic databases

Muhammad Muzammal · Rajeev Raman

Received: 11 April 2013 / Revised: 11 May 2014 / Accepted: 3 July 2014 / Published online: 24 July 2014

Abstract This paper considers the problem of *sequential pattern mining* (SPM) in *probabilistic databases*. Specifically, we consider SPM in situations where there is uncertainty in associating an *event* with a *source*, model this kind of uncertainty in the probabilistic database framework and consider the problem of enumerating all sequences whose *expected* support is sufficiently large. We give an algorithm based on dynamic programming to compute the expected support of a sequential pattern. Next, we propose three algorithms for mining sequential patterns from probabilistic databases. The first two algorithms are based on the candidate generation framework—one each based on a *breadth-first* (similar to GSP) and a *depth-first* (similar to SPAM) exploration of the search space. The third one is based on the pattern-growth framework (similar to PrefixSpan). We propose optimizations that mitigate the effects of the expensive dynamic programming computation step. We give an empirical evaluation of the probabilistic SPM algorithms and the optimizations and demonstrate the scalability of the algorithms in terms of CPU time and the memory usage. We also demonstrate the effectiveness of the probabilistic SPM framework in extracting meaningful sequences in the presence of noise.

Keywords Mining uncertain data · Sequential pattern mining · Probabilistic databases

1 Introduction

The *sequential pattern mining (SPM)* problem, or finding frequent sequences of events in data with a temporal component, was first introduced by Agrawal and Srikant [1] and has been studied extensively in the literature [2–5]. In classical SPM, it is assumed that the

M. Muzammal (✉)
Department of Computer Science, Bahria University, Islamabad, Pakistan
e-mail: muzammal@bui.edu.pk

R. Raman
Department of Computer Science, University of Leicester, Leicester, UK
e-mail: r.raman@mcs.le.ac.uk

data to be mined is deterministic. However, data obtained from many real-life applications is inherently noisy or uncertain [6]. *Probabilistic databases* is a popular framework for modelling uncertainties in data [7] and captures a wide range of applications [6]. Recently, several data mining and ranking problems have been studied in this framework, including top-k [8,9] and frequent itemset mining (FIM) [10–16]. This paper is the first to consider algorithms for uncertain SPM in the probabilistic databases framework (however, other kinds of uncertainty in SPM have been previously studied [17,18]).

In classical SPM, the event database 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 object (source) recorded by a camera (event). 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 the 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 [10,11,13]). In this work, we focus on uncertainty in the source which we refer to as *source-level uncertainty (SLU)*. Uncertainty in the source attribute could arise in situations such as:

(a) a customer (source) purchases some items (event) from a superstore and provides identity information, e.g. by filling a form. As the customer's details may be incomplete or imprecise, multiple matches may emerge in the customer database and thus uncertainty is introduced in the source attribute.
(b) a vehicle (source) is identified by a camera (event) using methods such as automatic number plate recognition (ANPR), which are inherently noisy. For example, a sample tuple in the `SIGHTING (time t, camera location l, vehicle Z)` relation that records vehicle sightings may look like `SIGHTING(103, p, {(Y81 UV: 0.6)(YB1 UV: 0.3)(Y81 UU:0.1)})` which means that a vehicle was recorded passing a camera at location p at time 103, but the number plate of the vehicle was uncertain, and could have been Y81 UV, YB1 UV or Y81 UU, with certainties 0.6, 0.3 and 0.1, respectively. To mine patterns such as "10 % of cars pass camera X, then camera Y and later camera Z", we consider each car as a source and each sighting as an event.

In such scenarios, it is certain that an event occurred (e.g. a customer bought some items, a vehicle passed a camera) but the source associated with that event is uncertain. The software which performs the matching would typically assign confidence values to the various alternative matches. A notable example of a large scale database gathering data using ANPR technology is the UK police database [19], and studies suggest that even the most advanced ANPR systems have only upto 90 % accuracy even under most suitable weather conditions [20]. We model the above scenarios by assuming that each event is associated with a probability distribution over possible sources that could have resulted in the event. This formulation thus shows attribute-level uncertainty in the source attribute [7].

In Muzammal and Raman [21], two measures of "frequentness", namely *expected support* and *probabilistic frequentness*, used for FIM in probabilistic databases [11,13] were adapted to SPM. It was shown that although probabilistic frequentness is more descriptive than expected support, even deciding if a sequence in an SLU database is probabilistically frequent is #P-complete [21]. This paper is focussed on efficient algorithms for SPM in an SLU database under the expected support measure. Our contributions are as follows:

1. We give a dynamic programming (DP) algorithm to determine efficiently the probability that a given source supports a sequence (the *source support probability*) and show that this is enough to compute the expected support of a sequence in an SLU database.
2. We propose two algorithms based on the candidate generation framework, one each based on a breadth-first and a depth-first exploration of the search space, and an algorithm based on the pattern-growth framework using the idea of projected databases, to find all frequent sequences in an SLU database according to the expected support criterion.
3. To speed up the support computation, we exploit properties of the DP to obtain algorithms for:
   (a) highly efficient computation of frequent 1-sequences, which allows us to compute all frequent 1-sequences in a (projected) database in linear time,
   (b) *incremental computation* of the DP matrix, which allows us to minimize the amount of time spent on the 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 scalability in terms of CPU time and memory usage, as well as the effectiveness of the above optimizations.
5. We empirically demonstrate the effectiveness of the probabilistic SPM framework at extracting meaningful sequences in the presence of noise.

1.1 Significance of results

We now highlight some key findings:

- 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 [10,12,13].
- In an SLU database, there are dependencies between different sources. For example, consider a sample SLU database having just one event e, but the source associated with this event could be one of sources X or Y, each with probability 0.5. Were these probabilities independent, the probability that e is associated with neither X nor Y would be $(1 - 0.5) \times (1 - 0.5) = 0.25$. However, since we are modelling the situation that e is guaranteed to have occurred, and there is no third source that could have given rise to e, it is impossible (i.e. probability 0) that e is associated with neither X nor Y. It is therefore slightly unexpected that despite this dependency, we can calculate the expected support efficiently (recall that determining probabilistic frequentness was shown to be intractable in [21]).
- The breadth-first and depth-first algorithms (2) have a high-level similarity to GSP [2] and SPADE/SPAM [3,5]. 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. The efficiency of our algorithms also depends 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 [12] for FIM in probabilistic databases, the SPM problem is more complex, and our pruning rule is harder to obtain.
- Although the pattern-growth approach (FP-tree) does not adapt well to the uncertain FIM [22], we show with the help of experiments that this is not the case for uncertain SPM—the pattern-growth approach (PrefixSpan [4]) when adapted to the uncertain case (using customized fast frequent 1-sequence computation and incremental support com-

putation), appears to be more scalable in terms of CPU time and memory usage than the candidate generation counterparts.

1.2 Related work

Classical SPM has been studied extensively [2–5]. Modelling uncertain data as *probabilistic* databases [6,7] has led to several ranking/mining problems being studied in this context. The top-k problem (a ranking problem) has been studied intensively (see [8,9,23] and references therein). In particular [9] highlights subtle issues in the semantics of the term "top-k" when applied to probabilistic databases.

FIM in probabilistic databases was studied under the *expected* support measure in [10,12,13] and under the *probabilistic frequentness* measure in [11,14]. As computing the probabilistic frequentness of an itemset in an uncertain database is a computationally expensive task, Calders et al. [16] and Wang et al. [15] propose approximating the probabilistic frequentness of an itemset. Sun et al. [14] proposed mining association rules from uncertain data. Tong et al. [22] give a comparison of different uncertain FIM algorithms in terms of CPU time and memory usage.

In probabilistic SPM, the individual attributes in a sequence of records, namely the time-stamp, the event and the source, or the sequence itself may be uncertain. To the best of our knowledge, apart from [21], the SPM problem in probabilistic databases has not been studied under current settings, i.e. considering uncertainty in the source attribute and using the expected support measure. Uncertainty in the time-stamp attribute was considered in [18]—we do not consider time to be uncertain. Muzammal and Raman [21] proposed two models of uncertainty, namely event-level uncertainty (ELU) and source-level uncertainty (SLU) and adapted two measures of frequentness, namely expected support and probabilistic frequentness from uncertain FIM to probabilistic SPM. Yang et al. [17] studies SPM in "noisy" sequences, but the model proposed there is very different to ours — it is similar to but less flexible than the ELU model in [21], as it does not allow us to express uncertainty in the source, the focus of this paper—and does not fit in the probabilistic database framework. Furthermore, the measure of support of a candidate sequence (the probability of the "best alignment") is very different from ours and does not have a natural "possible worlds" interpretation, particularly if used for SLU. Hooshadat et al. [24] considered uncertainty in the event (similar to the ELU model in [21]) and proposed UAPRIORI algorithm to compute frequent sequences using the expected support measure. Zhao et al. [25] considered uncertainty in the event (in a way similar to the ELU model in [21]) and the sequence, and proposed pattern-growth algorithms using the probabilistic frequentness measure. Probabilistic frequentness was shown to be #P-Complete for the SLU model by [21]. Wan et al. [26] considered mining frequent episodes (sub-sequences) from uncertain sequence data using the ELU model and the probabilistic frequentness measure, and proposed the probabilistic frequent serial episode mining problem and also an exact and an approximate solution for the problem. Achar et al. [27] proposed a pattern-growth-based algorithm for the probabilistic frequent serial episode mining problem.

2 Problem statement

We first review the classical SPM problem [1,2]. See Table 1 for a list of useful notation.

Table 1 A list of useful notation

Notation	Meaning
D^p	Probabilistic database
D^*	Possible world
$Sup(s, D^*)$	Support of a sequence s in a possible world D^*
$ES(s, D^p)$	Expected Support of a sequence s in D^p
$X_i(s, D)$	An indicator variable with value 1 if s is a sub-sequence of source sequence for source i, 0 otherwise
$\Pr(s \preceq D_i^p)$	Probability that s is a sub-sequence of source sequence for source i
$E(.)$	Expectation of a random variable
C_j	Candidate sequence of length j
L_j	Frequent sequence of length j
$D^{s,p}$	An s-projected database, where s is a sequence (and is a prefix)

2.1 Classical SPM

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 $Sup(s, D)$, as $\sum_{i=1}^{m} X_i(s, D)$.

The objective is to find all sequences s such that $Sup(s, D) \geq \theta$ for some user-defined threshold $0 \leq \theta \leq m$.

2.2 SLU database

We define an SLU 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 2.

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 SLU database of Table 2

Table 2 A sample SLU database

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)	$(X, 0.7)(Z, 0.3)$

Table 3 The complete set of possible worlds for the SLU database of Table 2 along with their probabilities

D^*	X	Y	Z	$\Pr(D^*)$	$Sup(s, D^*)$
D_1^*	$(a, d)(a, b)(b, c)$	$\langle\rangle$	(a)	0.126	1
D_2^*	$(a, d)(a, b)$	$\langle\rangle$	$(a)(b, c)$	0.054	2
D_3^*	$(a, d)(b, c)$	(a, b)	(a)	0.084	1
D_4^*	(a, d)	(a, b)	$(a)(b, c)$	0.036	1
D_5^*	$(a, d)(b, c)$	$\langle\rangle$	$(a)(a, b)$	0.210	2
D_6^*	(a, d)	$\langle\rangle$	$(a)(a, b)(b, c)$	0.090	1
D_7^*	$(a, b)(b, c)$	(a, d)	(a)	0.084	1
D_8^*	(a, b)	(a, d)	$(a)(b, c)$	0.036	1
D_9^*	(b, c)	$(a, d)(a, b)$	(a)	0.056	1
D_{10}^*	$\langle\rangle$	$(a, d)(a, b)$	$(a)(b, c)$	0.024	2
D_{11}^*	(b, c)	(a, d)	$(a)(a, b)$	0.140	1
D_{12}^*	$\langle\rangle$	(a, d)	$(a)(a, b)(b, c)$	0.060	1

The right-most column shows the support of the sequence $s = \langle(a)(b)\rangle$ in each possible world

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 $\Pr[D^*] = 0.6 \times 1.0 \times 0.3 \times 0.7 = 0.126$. The complete set of possible worlds for the SLU database of Table 2 is shown in Table 3. 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:

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

The problem we consider is:

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

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 dynamic programming.

3 Computing expected support

p-sequence. A *p-sequence* is analogous to a source sequence in classical SPM and is a sequence of the form $\langle(e_1, c_1) \ldots (e_k, c_k)\rangle$, where e_j is an event and c_j is a confidence value.

Table 4 The SLU database of Table 2 transformed to p-sequences (R)

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)	$(X, 0.7)(Z, 0.3)$

$\Rightarrow$

	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	$(a : 1.0)(a, b : 0.5)(b, c : 0.3)$

Note that in the p-sequences representation, the event e_1 (marked with †) must be associated with exactly one of the sources X and Y in any possible world

In examples, we write a p-sequence $\langle(\{a, d\}, 0.4), (\{a, b\}, 0.2)\rangle$ as $\langle(a, d : 0.4)(a, b : 0.2)\rangle$. An SLU database D^p can be viewed as a collection of p-sequences $D^p_1, \ldots, D^p_m$, where D^p_i 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 4(R)).

However, the p-sequences corresponding to different sources are *not* independent, as illustrated in Table 4(R). Thus, one may view an SLU database as a collection of p-sequences with dependencies in the form of x-tuples [9]. Despite the dependencies, we show that we can still process the p-sequences independently for the purpose of expected support computation:

Lemma 3.1 *Given an SLU database D^p in the form of p-sequences and a sequence s, we can compute the expected support of s in D^p by computing the source support probability for each source i independently, that is $ES(s, D^p) = \sum_{i=1}^{m} \Pr(s \preceq D^p_i)$.*

Proof Any possible world D^* of D^p is a deterministic database. Therefore, the support of s in D^*, denoted by $Sup(s, D^*)$, equals $\sum_{i=1}^{m} X_i(s, D^*)$, where $X_i(s, D^*)$ is the indicator variable whose value is 1 if source σ_i supports s in D^* and 0 otherwise (Sect. 2). From Eq. (1), we get that:

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

where (abusing notation slightly) we have introduced a new indicator *random* variable $X_i(s, D^p)$, whose value is 1 in all those possible worlds where source σ_i supports s, and 0 in all other possible worlds, and $E(\cdot)$ denotes the expected value of a random variable. Since X_i is a 0-1 variable,

$$\mathrm{E}[X_i(s, D^p)] = \Pr[s \preceq D^p_i], \tag{3}$$

where the right-hand quantity in Eq. (3) is the probability that source i supports a sequence s, and is referred to henceforth as the *source support probability*.

Thus, from Eqs. (2) and (3), we show that:

$$ES(s, D^p) = \textstyle\sum_{i=1}^{m} \Pr[s \preceq D^p_i], \tag{4}$$

which proves the lemma. □

We now give an example to explain Lemma 3.1. Consider a sequence $s = \langle(a)(b)\rangle$ and the probabilistic database D^p of Table 4. The probability that source X supports s is the sum of probabilities of all the possible worlds where X supports s, i.e. the possible worlds D^*_1, D^*_2, D^*_3, D^*_5 and D^*_7, and thus $\Pr[s \preceq D^p_X] = 0.126 + 0.054 + 0.084 + 0.210 + 0.084 = 0.558$.

Table 5 Computing $\Pr[\langle(a)(b)\rangle \preceq D_X^p]$ using DP in the SLU database of Table 4

		$(a, d : 0.6)$	$(a, b : 0.3)$	$(b, c : 0.7)$
	A[0,0] = 1	A[0,1] = 1	A[0,2] = 1	A[0,3] = 1
$\langle(a)\rangle$	A[1,0] = 0	A[1,1] = $0.4 \times 0 + 0.6 \times 1 = 0.6$	A[1,2] = $0.7 \times 0.6 + 0.3 \times 1 = 0.72$	A[1,3] = 0.72
$\langle(a)(b)\rangle$	A[2,0] = 0	A[2,1] = 0	A[2,2] = $0.7 \times 0 + 0.3 \times 0.6 = 0.18$	A[2,3] = $0.3 \times 0.18 + 0.7 \times 0.72 = 0.558$

In Table 5, the value A[2,3] is the probability that the sequence $\langle(a)(b)\rangle$ is supported by source X

Similarly, $\Pr[s \preceq D_Y^p]$ and $\Pr[s \preceq D_Z^p]$ are computed as 0.08 and 0.035, respectively. Thus, $\Pr[s \preceq D^p] = 0.058 + 0.08 + 0.035$, the same value obtained directly from Eq. (1).

Although Lemma 3.1 reduces the computation of expected support to computing source support probabilities, the latter can still not be done naively: e.g. if $D_i^p = \langle(a, b : c_1)(a, b : c_2)\ldots(a, b : c_q)\rangle$, then there are $O(q^{2k})$ ways in which a sequence $s = \langle\underbrace{(a)(a, b)\ldots(a)(a, b)}_{k \text{ times}}\rangle$ could be supported by source i. In the next section, we give an algorithm that uses DP to compute the source support probability.

3.1 Source support probability

We now discuss how to compute $\Pr[s \preceq D_i^p]$. Given a p-sequence $D_i^p = \langle(e_1, c_1), \ldots, (e_r, c_r)\rangle$ and a sequence $s = \langle s_1, \ldots, s_q\rangle$, 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 $1 \le k \le q$ and $1 \le \ell \le r$, $A[k, \ell]$ will contain $\Pr[\langle s_1, \ldots, s_k\rangle \preceq \langle(e_1, c_1), \ldots, (e_\ell, c_\ell)\rangle]$. For example, the cell $A[2, 3]$ in Table 5 contains the value $\Pr[\langle(a)(b)\rangle \preceq D_X^p]$, i.e. the probability that the sequence $\langle(a)(b)\rangle$ is supported by the source sequence of source X. We set $A[0, \ell] = 1$ for all ℓ, $0 \le \ell \le r$ and $A[k, 0] = 0$ for all $1 \le k \le q$, and compute the other values row-by-row. For $1 \le k \le q$ and $1 \le \ell \le r$, define:

$$c_{k\ell}^* = \begin{cases} c_\ell & \text{if } s_k \subseteq e_\ell \\ 0 & \text{otherwise} \end{cases} \tag{5}$$

The interpretation of Eq. (5) 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:

$$A[k, \ell] = (1 - c_{k\ell}^*) * A[k, \ell - 1] + c_{k\ell}^* * A[k - 1, \ell - 1]. \tag{6}$$

The reason Eq. (6) 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. 6) and those where it is (the second term in Eq. 6).

Lemma 3.2 *Given a p-sequence D_i^p and a sequence s, by applying Eq. (6) repeatedly, we correctly compute* $\Pr[s \preceq D_i^p]$.

Table 5 shows the computation of the source support probability of an example sequence $s = \langle(a)(b)\rangle$ for source X in the SLU database of Table 4. Similarly, we can compute $\Pr[s \preceq D_Y^p] = 0.08$ and $\Pr[s \preceq D_Z^p] = 0.35$, so the expected support of $\langle(a)(b)\rangle$ in the SLU

database of Table 4 is $0.558 + 0.08 + 0.35 = 1.288$, the same value obtained by the direct application of Eq. (1).

4 Candidate generation

We now describe probabilistic SPM algorithms based on the candidate generation framework. We describe two candidate generation approaches based on a breadth-first (BFS) and a depth-first (DFS) exploration of the search space. Our proposed BFS approach is similar to GSP [2], and our DFS approach is similar to the depth-first variant of SPADE [3] (SPADE has a breadth-first variant as well) or SPAM [5], in search space exploration.

Although the DP algorithm proposed in Sect. 3 computes the expected support of a sequence in an SLU database in polynomial rather than exponential time, the support computation is performed repeatedly as there could potentially be a large number of candidate sequences which need to be tested for being frequent. We first give some optimizations to speed up the support computation task.

4.1 Optimization

In this section, we describe two optimizations for the DP step:

1. *Fast Frequent 1-sequence Computation:* that is, computing all frequent 1-sequences in a single scan of the database in linear time.
2. *Incremental Support Computation:* that is, reusing the already computed results for DP.

Further, we show two properties of expected support that can be used to prune the search space.

3. *Apriori Pruning:* that is, pruning a candidate sequence if any of its subsequences is not frequent.
4. *Probabilistic Pruning:* that is, eliminating potential infrequent candidate sequences without support computation.

We now elaborate on each of these below.

4.1.1 Fast frequent 1-sequence computation

The DP algorithm can be used to find all frequent 1-sequences by applying Lemma 3.2 and Eq. (2). However, a naive approach, e.g. for each 1-sequence $\langle(x)\rangle$, $x \in \mathcal{I}$, computing $ES(\langle(x)\rangle, D^p)$ by applying Lemma 3.2 and Eq. (2), will require $|\mathcal{I}|$ scans of the database. In classical SPM, the task of finding all frequent 1-sequences can be performed in linear time in a single scan over the database. We show that we can achieve the same, i.e. find all frequent 1-sequences in linear time in a single scan over the database, in the probabilistic case as well.

We first give a simple intuitive closed-form expression for the source support probability of a 1-sequence. Given a 1-sequence $s = \langle(x)\rangle$, $x \in \mathcal{I}$, and a p-sequence D_i^p of length r, the probability with which source i supports s can be computed as:

$$\Pr[s \preceq D_i^p] = 1 - \prod_{\ell=1}^{r} (1 - c_{1\ell}^*). \tag{7}$$

The term on the right in Eq. (7) is the complement of the probability that s is not supported by D_i^p, which in other words, is the probability with which source i supports s. It is easy to verify by induction that using Eq. (7) gives the same answer as we get by applying Lemma 3.2.

For example, consider a 1-sequence $\langle(a)\rangle$ and the p-sequence for source X. $\Pr[\langle(a)\rangle \preceq D_X^p]$ is computed as $1-[(1-0.6)*(1-0.3)*(1-0.0)] = 0.720$, using Eq. (7); the same as obtained by Eq. (1) directly.

We now describe a procedure for computing the expected support of all 1-sequences. Initialize two arrays F and G, each of size $|\mathcal{I}|$, 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 $F[x] := (1-c_k)$ if $F[x] = 0$, and setting $F[x] := F[x]*(1-c_k)$ otherwise. It is clear that for all x such that $F[x] \neq 0$, $F[x] = \prod_{\ell=1}^{r}(1-c_{1\ell}^*)$, and so $1-F[x]$ is the source support probability for the sequence $\langle(x)\rangle$. For all x such that $F[x] \neq 0$ we set $G[x] = G[x] + 1 - F[x]$, and then reset $F[x]$ to zero (we use a linked list to keep track of non-zero entries in F rather than scanning F). At the end, for any 1-sequence $s = \langle(x)\rangle$, where $x \in \mathcal{I}$, $G[x] = ES(s, D^p)$.

4.1.2 Incremental support computation

Let s and t be two sequences. Following Ayres et al. [5], we 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), and 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 $s = \langle(a)(b,c)\rangle$ and $x = d$, S- and I-extensions of s are $\langle(a)(b,c)(d)\rangle$ and $\langle(a)(b,c,d)\rangle$, respectively. Similar to classical SPM, for an existing frequent sequence s, we generate candidate sequences t that are either S- or I-extensions s, and compute $ES(t, D^p)$ by computing $\Pr[t \preceq D_i^p]$ for all sources i. Whilst 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. For example, if $s = \langle(a)(b,c)\rangle$ and $t = \langle(a)(b,c)(d)\rangle$ (an S-extension of s), the DP rows for $\langle(a)\rangle$, $\langle(a)(b)\rangle$ and $\langle(a)(b,c)\rangle$ would be the same for any source i (the same holds for I-extensions of s) and therefore, could be reused in support computation.

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 4.1 *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 events 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. 6).

Now, consider the case where t is an 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-th 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, that is, 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

Algorithm 1 Incremental Support Computation (I-extension case)

1: $B_{i,t}[0] = 0$
2: **for all** $\ell = 1, \ldots, r$ **do**
3: **if** $t_q \not\subseteq e_\ell$ **then**
4: $B_{i,t}[\ell] = B_{i,t}[\ell - 1]$
5: **else**
6: $B_{i,t}[\ell] = (1 - c_\ell) * B_{i,t}[\ell - 1] + (B_{i,s}[\ell] - B_{i,s}[\ell - 1] * (1 - c_\ell))$

Table 6 Example illustrating the incremental support computation of $B_{i,t}$ for $t = \langle (a)(b, c) \rangle$ from $B_{i,s}$ where $s = \langle (a)(b) \rangle$, by computing $\Pr[t \preceq D_X^p]$ in the database of Table 2

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

Note that the row corresponding to $\langle (a) \rangle$ is *not* available

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 $A_{i,t}[q, \ell] = B_{i,t}[\ell]$. □

The pseudo-code for incremental support computation (I-extension case) is given in Algorithm 1, and an example of this computation is given in Table 6.

4.1.3 Apriori pruning

We note that an apriori property holds in the probabilistic setting as well, which is used similarly to classical SPM to prune the search space.

Lemma 4.2 *Given two sequences s and t, and an SLU database D^p, if s is a subsequence of t, then $ES(s, D^p) \geq ES(t, D^p)$.*

Proof For any two sequence s and t where s is a subsequence of t, we know by the apriori property that for all possible worlds $D^* \in PW(D^p)$, $Sup(s, D^*) \geq Sup(t, D^*)$, and from Eq. (1) we have,

$$\begin{aligned} ES(s, D^p) &= \textstyle\sum_{D^* \in PW(D^p)} \Pr[D^*] * Sup(s, D^*) \\ &\geq \textstyle\sum_{D^* \in PW(D^p)} \Pr[D^*] * Sup(t, D^*) \\ &= ES(t, D^p) \end{aligned}$$

□

4.1.4 Probabilistic pruning

We now describe a pruning technique that allows us to eliminate potential infrequent candidate sequences s without fully computing the expected support of s in D^p. For each source i,

we obtain an upper bound on $\Pr[s \preceq D_i^p]$, the probability with which source i supports s, and add up all the upper bounds; if the sum is below the threshold, s can be pruned. We first show:

Lemma 4.3 *Let $s = \langle s_1, \ldots, s_q \rangle$ be a sequence, and let D_i^p be a p-sequence. Then:*

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

Proof Let $A = A_{i,s}$ be the DP matrix for computing $\Pr[s \preceq D_i^p]$. For $\ell = 0, \ldots, r$, let $p_\ell = \Pr[\langle s_q \rangle \preceq \langle (e_1, c_1), \ldots, (e_\ell, c_\ell) \rangle]$; p_r therefore is precisely $\Pr[\langle s_q \rangle \preceq D_i^p]$. 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. (5). We now prove by induction on ℓ that:

$$A[q, \ell] \leq A[q-1, \ell] * p_\ell, \tag{8}$$

substituting $\ell = r$ in Eq. (8) proves the lemma. We now prove Eq. (8), which clearly holds for $\ell = 0$ since $A[q, 0] = A[q-1, 0] = p_0 = 0$. Subsequently:

$$\begin{aligned}
A[q, \ell] &= (1 - c^*_{q\ell}) * A[q, \ell - 1] + c^*_{q\ell} * A[q-1, \ell - 1] \quad \text{(Eq. 6)} \\
&\leq (1 - c^*_{q\ell}) * A[q-1, \ell - 1] * p_{\ell-1} + c^*_{q\ell} * A[q-1, \ell - 1] \quad \text{(Eq. 8)} \\
&\leq A[q-1, \ell - 1] * \left((1 - c^*_{q\ell}) * p_{\ell-1} + c^*_{q\ell} \right) \\
&\leq A[q-1, \ell - 1] * p_\ell \\
&\leq A[q-1, \ell] * p_\ell.
\end{aligned}$$

□

We now indicate how Lemma 4.3 is used. Suppose, for example, that we have a candidate sequence $s = \langle (a)(b, c)(a) \rangle$, and a source X. By Lemma 4.3:

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

Observe that the quantities on the RHS are computed as intermediate results during the fast frequent 1-sequence computation, and can be saved in a small data structure associated with each source. Of course, if $\Pr[\langle (a)(b, c) \rangle \preceq D_X^p]$ is available, an even tighter bound is $\Pr[\langle (a)(b, c) \rangle \preceq D_X^p] * \Pr[\langle (a) \rangle \preceq D_X^p]$.

4.2 Breadth-first exploration

An overview of our BFS approach is in Algorithm 2. We now describe some details. Each execution of lines (6)–(11) is called a *phase*. In the j-th phase, C_j is the set of candidate j-sequences and L_j is the set of frequent j-sequences. Line 4 is performed using the fast frequent 1-sequence computation (see Sect. 4.1.1). Line 6 is done as in [2]: 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, $\langle (a)(b, c)(d) \rangle$ joins with $\langle (b, c)(d, e) \rangle$ to yield $\langle (a)(b, c)(d, e) \rangle$ and $\langle (a, b)(c, d) \rangle$ joins with $\langle (b)(c, d)(e) \rangle$ to yield $\langle (a, b)(c, d)(e) \rangle$. The join of two 1-sequences $\langle (a) \rangle$ and $\langle (b) \rangle$, results in both $\langle (a)(b) \rangle$ and $\langle (a, b) \rangle$ being output.

Algorithm 2 Breadth-First Exploration

1: **Input:** SLU probabilistic database D^p and support threshold θ.
2: **Output:** All sequences s with $ES(s, D^p) \geq \theta$.
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 $ES(s, D^p)$ (using DP (Sect. 3))
10: $L_{j+1} \leftarrow$ all sequences $s \in C_{j+1}$ s.t. $ES(s, D^p) \geq \theta$}.
11: $j \leftarrow j + 1$
12: Stop and output $L_1 \cup \ldots \cup L_j$

Algorithm 3 Depth-First Exploration

1: **Input:** SLU probabilistic database D^p and support threshold θ.
2: **Output:** All sequences s with $ES(s, D^p) \geq \theta$.
3: $L \leftarrow \emptyset, L_x \leftarrow \emptyset$
4: $L_1 \leftarrow$ ComputeFrequent-1(D^p)
5: **for all** sequences $x \in L_1$ **do**
6: $L_x \leftarrow$ Call TraverseDFS(x, L_1)
7: $L \leftarrow L \cup L_x$
8: Output all sequences in L

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 $s \in N_{i,j+1}$, 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.

Given $N_{i,j+1}$ we now discuss computing the support of a sequence t in source i after we have computed the support of a sequence s. 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}.

4.3 Depth-first exploration

An overview of our depth-first approach is in Algorithm 3. We first compute L_1, the set of frequent 1-sequences (line 4), and assume that L_1 is in ascending order. We then explore the pattern sub-lattice (lines 5–6), the details of which are given in Algorithm 4, and report all frequent sequences (line 8).

Algorithm 4 Depth-First Traversal

1: **Input:** SLU probabilistic database D^p and a sequence s.
2: **Output:** All possible extensions of s with $ES(s, D^p) \geq \theta$.
3: **function** TraverseDFS(s, L_1)
4: $L \leftarrow \emptyset$
5: **for all** valid $x \in L_1$ **do**
6: $t \leftarrow \langle s \cdot \{x\} \rangle$ {S-extension}
7: **if** t is not pruned **then**
8: Compute $ES(t, D^p)$
9: **if** $ES(t, D^p) \geq \theta$ **then**
10: $L \leftarrow L \cup t$
11: TraverseDFS(t, L_1)
12: **else**
13: Mark $\{x\}$ as invalid S-extension item.
14: $t \leftarrow \langle s_1, \ldots, s_q \cup \{x\} \rangle$ {I-extension}
15: **if** t is not pruned **then**
16: Compute $ES(t, D^p)$
17: **if** $ES(t, D^p) \geq \theta$ **then**
18: $L \leftarrow L \cup t$
19: TraverseDFS(t, L_1)
20: **else**
21: Mark $\{x\}$ as invalid I-extension item.
22: **return** L
23: **end function**

Consider a call of TraverseDFS(t, L_1) (Algorithm 4), where t is some k-sequence and is obtained by appending a frequent item x to s, either as an S-extension of s or as an I-extension of s. Prior to the support computation, we check all lexicographically smaller $(k-1)$-subsequences of t (which would have been explored previously) for frequentness, and reject t as infrequent if this test fails (lines 7 and 15). We can then apply probabilistic pruning to t, and if t is still not pruned, we compute its support (lines 8 and 16). If at any stage, t is pruned, or if it is found to be infrequent, 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 t (lines 13 and 21), as in [5]. 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 Sect. 4.1.2) 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.

5 Pattern growth

We now give a pattern-growth algorithm similar to PrefixSpan [4] for enumerating all frequent sequences. We first give a few definitions.

Definition 5.1 (*weak-prefix*) Given a sequence $s = \langle s_1, \ldots, s_q \rangle$ and a p-sequence $t = \langle (t_1, c_1), \ldots, (t_r, c_r) \rangle$, s is called a weak-prefix of t if there exist indices $1 \leq j_1 < j_2 < \ldots < j_q \leq r$ such that (1) $s_i \subseteq t_{j_i}$, for all $1 \leq i \leq (q-1)$, and for all k, $j_i < k < j_{i+1}$,

$s_{i+1} \not\subseteq t_k$, and (2) $s_q \subseteq t_{j_q}$, and all the items in $t_{j_q} - s_q$ are lexicographically after those in s_q.

Definition 5.2 (*weak-suffix*) Given a sequence $s = \langle s_1, \ldots, s_q \rangle$ and a p-sequence $t = \langle (t_1, c_1), \ldots, (t_r, c_r) \rangle$, where s is a weak-prefix of t, a p-sequence $u = \langle (u_q, c_q), \ldots, (u_r, c_r) \rangle$ is the weak-suffix of t with respect to s, where $u_k = (t_k - s_k)$ for $k = q$, and $u_k = t_k$, for $k = q + 1, \ldots, r$.

For example, for a p-sequence $s = \langle (a : 0.3)(a, b, d : 0.8)(b, c : 0.4)(d, e : 0.5) \rangle$, sequences such as $\langle (a) \rangle$ and $\langle (a)(b, c) \rangle$ are weak-prefixes of s, whereas $\langle (a)(c, d) \rangle$ and $\langle (a)(c)(b) \rangle$ are not. For the weak-prefixes $\langle (a) \rangle$ and $\langle (a)(b) \rangle$, the weak-suffixes of s are $\langle (a, b, d : 0.8)(b, c : 0.4)(d, e : 0.5) \rangle$ and $\langle (-, d : 0.8)(b, c : 0.4)(d, e : 0.5) \rangle$, respectively, where '$-$' means that one or more items in the event are part of the weak-prefix. In what follows, we sometimes use prefix and suffix rather than weak-prefix and weak-suffix when it is clear from the context.

According to [4], the set of all frequent sequential patterns can be partitioned into as many subsets as the number of frequent 1-sequences. If $\{\langle x_1 \rangle, \ldots, \langle x_n \rangle\}$ are all 1-sequences, the i-th subset is prefixed with $\langle x_i \rangle$, $1 \leq i \leq n$. Then, based on each prefix, each subset of sequential patterns could be further subdivided recursively. To mine the subset of sequential patterns based on a prefix, the projected database corresponding to that prefix need to be constructed. We first give the definition of a projected database in the probabilistic case.

Definition 5.3 (*Projected Database*) Given a probabilistic database D^p in the form of p-sequences $D_1^p, \ldots, D_m^p$, and a sequence s, an s-projected probabilistic database $D^{s,p}$ is the collection of weak-suffixes of the p-sequences in D^p with respect to the weak-prefix s.

For example, consider a sample probabilistic database having two p-sequences $\{\langle (a : 0.4)(b, c : 0.7)(a, d, e : 0.3) \rangle, \langle (a, b : 0.9)(a, b, d : 0.8)(d, e : 0.3) \rangle\}$. For a sequence $\langle (a) \rangle$ as a weak-prefix, an $\langle (a) \rangle$-projected database is $\{\langle (b, c : 0.7)(a, d, e : 0.3) \rangle, \langle (-, b : 0.9)(a, b, d : 0.8)(d, e : 0.3) \rangle\}$.

PrefixSpan works as follows. It first finds the complete set of frequent 1-sequences. Next, the set of projected databases is constructed prefixed with each frequent 1-sequence. Then, in each projected database, it again finds the set of frequent 1-sequences local to that projected database, and keeps on building and mining the projected databases this way, recursively.

It appears that based on the definitions of weak-prefix (Definition 5.1) and projected database (Definition 5.3), corresponding projected databases could be constructed and using the fast frequent 1-sequence computation (Sect. 4.1), the complete set of sequential patterns could be obtained. However, in the probabilistic setting, it is not correct to simply perform the fast frequent 1-sequence computation in a projected database. For example, consider a sample probabilistic database having two source sequences $\{\langle (a : 0.5)(b : 0.5)(b : 0.5)(a : 0.5) \rangle, \langle (a : 0.5)(b : 0.5)(a : 0.5)(b : 0.5) \rangle\}$. An $\langle (a) \rangle$-projected database will contain two suffixes $\{\langle (b : 0.5)(b : 0.5)(a : 0.5) \rangle, \langle (b : 0.5)(a : 0.5)(b : 0.5) \rangle\}$. When considering whether $\langle (a)(b) \rangle$ is frequent, it is not correct to compute the expected support of (b) in the projected database. For example, both suffixes above would give the same contribution (0.75) to the support of (b) in the projected database, but clearly their support for $\langle (a)(b) \rangle$ is different. The second sequence can form $\langle (a)(b) \rangle$ in three ways: the first a, implicit in the weak-prefix, with either of the two b's, or the second a with the second b, so its contribution to the support of $\langle (a)(b) \rangle$ is 0.75. The first sequence can form $\langle (a)(b) \rangle$ in only two ways, and its contribution is 0.5. See Table 7 for an example of this computation.

We now show that we can use the DP algorithm along with the fast frequent 1-sequence computation to find all frequent sequences using the pattern-growth framework.

Algorithm 5 Pattern-Growth Algorithm

1: **Input:** SLU probabilistic database D^p and support threshold θ.
2: **Output:** All sequences s with $ES(s, D^p) \geq \theta$.
3: $L_1 \leftarrow$ rm ComputeFrequent-1-sequences(D^p)
4: **for all** sequences $x \in L_1$ **do**
5: Compute $B_{i,x}$ arrays
6: Call ProjectedDB(x, $D^{s,p}$)
7: **function** ProjectedDB(s, $D^{s,p}$)
8: $L_S \leftarrow$ Compute Frequent S-extensions
9: $L_I \leftarrow$ Compute Frequent I-extensions
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, $D^{t,p}$)
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, $D^{t,p}$)
19: **end function**

5.1 Pattern-growth algorithm

An overview of our pattern-growth algorithm is in Algorithm 5. Assuming that the probabilistic database D^p contains only the frequent items, we first compute the set of frequent 1-sequences L_1. Assume L_1 is in ascending order. For each 1-sequence $\langle x \rangle$, we first create an $\langle x \rangle$-projected database $D^{x,p}$, and also compute the $B_{i,x}$ arrays for each suffix i in $D^{x,p}$ and then call the ProjectedDB(x, $D^{x,p}$) sub-routine recursively. In the ProjectedDB(s, $D^{s,p}$) sub-routine, we compute all the frequent S- and I-extensions of s using a modification of fast frequent 1-sequence computation. We call the computation of all S- and I-extensions of a sequence s the pattern-growth step, and elaborate on it in the coming section. Then, for every sequence t which is a frequent S- or I-extension of s, we create a $\langle t \rangle$-projected database $D^{t,p}$ and also compute $B_{i,t}$ arrays, and call the ProjectedDB(t, $D^{t,p}$) sub-routine recursively to mine all frequent sequential patterns. We now elaborate on the pattern-growth step.

5.2 Pattern-growth step

Pre-conditions:

1. $s = \langle s_1, \ldots, s_q \rangle$ is a previously discovered frequent sequence.
2. An $\langle s \rangle$-projected database $D^{s,p}$ is available.
3. The $B_{i,s}$ arrays for all sources i in $D^{s,p}$ are also available.

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

5.2.1 S-extension computation

We first compute the frequent S-extensions of s as follows. Initialize two arrays F and G, each of size $|\mathcal{I}|$, to zero. Recall that an $\langle s \rangle$-projected database $D^{s,p}$ is a collection of suffixes of s in D^p, where a suffix is of the form $\langle (u_q, c_q), \ldots, (u_r, c_r) \rangle$. As the $B_{i,s}$ arrays have

Table 7 An example of computing the expected support of all S-extensions of $s = \langle(a)\rangle$ for a sample p-sequence $D_i^p = \langle(a, d, e : 0.6)(b, c : 0.3)(a, b, c : 0.7)(c, d, e : 1.0)\rangle$

D_i^p	$(a, d, e : 0.6)$	$(b, c : 0.3)$	$(a, b, c : 0.7)$	$(c, d, e : 1.0)$
$B_{i,s}$	0.60	0.60	0.88	0.88
$\langle(a)\rangle$	0.000	0.000	$(1 - 0.7) * 0 + (0.7 * 0.6) = 0.420$	0.420
$\langle(b)\rangle$	0.000	$(1 - 0.3) * 0 + (0.3 * 0.6) = 0.180$	$(1 - 0.7) * 0.180 + (0.7 * 0.6) = 0.474$	0.474
$\langle(c)\rangle$	0.000	$(1 - 0.3) * 0 + (0.3 * 0.6) = 0.180$	$(1 - 0.7) * 0.180 + (0.7 * 0.6) = 0.474$	$(1 - 1) * 0.474 + (1 * 0.88) = 0.880$
$\langle(d)\rangle$	0.000	0.000	0.000	$(1 - 1) * 0.0 + (1 * 0.88) = 0.880$
$\langle(e)\rangle$	0.000	0.000	0.000	$(1 - 1) * 0.0 + (1 * 0.88) = 0.880$
	(i)	(ii)	(iii)	(iv)

The cells in columns labelled (i)–(iv) show the entries in F array after each event in D_i^p is processed. The values in the column (iv) are updated to G

already been computed, we consider each suffix of s in $D^{s,p}$ in turn, and for every item x in u_k, for $k = q + 1, \ldots, r$, update $F[x]$ as follows:

$$F[x] := ((1 - c_k) * F[x]) + (c_k * B_{i,s}[k - 1]). \tag{9}$$

We keep track of all the non-zero entries in $F[x]$, and once we are finished with a suffix, we update $G[x] := G[x] + F[x]$ and reset $F[x]$ to zero. After all the suffixes of s in $D^{s,p}$ have been processed, all the entries in $G[x] \geq \theta$ correspond to frequent S-extensions of s. An example of this computation is shown in Table 7.

5.2.2 I-extension computation

For the I-extensions case, we use the same arrays F and G as in the S-extensions case. Given the suffixes of s in $D^{s,p}$, we consider every suffix of s in turn. As a suffix is of the form $u = \langle(u_q, c_q), \ldots, (u_r, c_r)\rangle$, the possible I-extensions of s in u could be the items in $u_k - s_k$ for $k = q, \ldots, r$, where $s_k \subseteq u_k$ and all the items in $u_k - s_k$ are lexicographically after those in s_k. Considering every event u_k in u in turn where $s_k \subseteq u_k$, for $k = q, \ldots, r$, we update $F[x]$ for every item x lexicographically after those in s_k, in $u_k - s_k$ as follows:

$$F[x] := (1 - c_k) * F[x] + (B_{i,s}[k] - B_{i,s}[k - 1] * (1 - c_k)). \tag{10}$$

We repeatedly update G similar to the S-extensions case, and after all the suffixes of s in $D^{s,p}$ have been processed, all the entries in G such that $G[x] \geq \theta$ correspond to frequent I-extensions of s.

6 Empirical evaluation

We now report on an empirical evaluation of our algorithms. We study the usefulness of probabilistic pruning, and also evaluate the scalability, memory usage, and relative performance of our algorithms. Finally, we evaluate the effectiveness of probabilistic SPM framework in extracting useful information in the presence of noise.

6.1 Experimental setup

We begin by describing the datasets used for experiments, and the SLU data generation.

6.1.1 Datasets

We use both real and synthetic datasets in our experiments. Note that these are all deterministic datasets which we transform to the probabilistic form as described below ([10,11,13–15] also used this approach). The real dataset *gazelle* is from Blue Martini Software and was used for KDD-CUP'2000 [28]. The dataset is the web click stream data of a webstore (gazelle.com), and contains a total of 29,369 customer (source) sequences. A customer may have multiple sessions associated with it, and a session may contain multiple page visits. The click stream information contains the session start/end time, and also the page visit date/time and the sequence number. Therefore, all the page visits by a customer can be transformed to a single click stream of page visits (source sequence), ordered by a time-stamp. There are a total of 1,423 distinct web pages, 35,722 sessions and 87,546 page visits. For more details see Kohavi et al. [28].

The synthetic datasets are generated using the IBM Quest data generator [1]. In our experiments, we fix the number of items[1] $N = 2\text{K}$ and set the rest of the parameters except the number of source sequences D and the average number of events per source sequence C, to default values. The default values for the remaining IBM Quest parameters are as follows: average number of items per itemset $T = 2.5$, number of potential frequent itemsets $N_I = 5000$, number of potential frequent sequences $N_S = 100$, average number of itemsets per frequent sequence $S = 7$ and average number of items per itemset in a frequent sequence $I = 2$.

We follow the naming convention of Zaki [3]: a dataset named `CiDjK` means that the average number of events per source sequence is i and the number of source sequences is j (in thousands). For example, the dataset `C10D20K` has on average 10 events per source sequence and 20K source sequences. As the rest of the parameters are either fixed or set to default values, we do not mention these parameters explicitly hereafter.

6.1.2 SLU data generation

We transform the above mentioned deterministic datasets to the probabilistic form as follows. Recall that a deterministic database $D = \langle r_1, \ldots, r_n \rangle$, is an ordered list of tuples of the form (eid, e, σ), where eid is an event-id (including a time-stamp), and e is the event which is associated with a source σ (Sect. 2). In an SLU database D^p, the source attribute σ is replaced by a probability distribution W over sources (Sect. 2).

In order to introduce uncertainty into our deterministic database D, we choose a *noise* parameter $\delta \in [0, 1]$, which controls the degree of uncertainty about a tuple r_i being associated with the same source j in D^p as it was in D. Whilst generating an SLU database, we set the number of sources in W to be at most two, unless stated otherwise.

Specifically, whilst generating an SLU database D^p from D, we process every tuple r_i in D in turn as follows. We generate a value $p \in [0, 1]$ from a Poisson distribution with a mean value δ: in the SLU database D^p, the tuple r_i is associated with the same source j as it was in D with a probability $1 - p$, and is associated with some other randomly chosen source

[1] In this section, we use parameter names derived from [1] rather than those introduced in Sect. 2 for consistency with previous work. For example, we use N rather than q to denote the number of items.

$\sigma_k \in \mathcal{S}, k \neq j$, with probability p. Note that if the value of δ is low, there is a greater chance that r_i is associated with the same source j in D^p as it was in D. In our experiments, we set $\delta = 10\,\%$, unless stated otherwise. The p-sequences in the SLU database D^p are the source sequences of probabilistic events ordered by a time-stamp.

In what follows, we use the term "synthetic dataset" for a dataset generated using IBM Quest data generator and then, transformed to SLU form and similarly, "real dataset" refers to `gazelle` transformed to SLU form. Further, in our experiments, we express the support threshold θ as a percentage of number of source sequences D, rather than an absolute number $0 \leq \theta \leq m$, for convenience' sake.

6.1.3 Experiments outline

We implemented the above algorithms in C# (Visual Studio .Net 2010), executed on a machine with a 3.2GHz Intel CPU and 3GB RAM running Microsoft Windows XP (SP3). To obtain the results (running time, memory usage or others), we generate three probabilistic instances of each deterministic dataset, and run each of our algorithms on every probabilistic instance several times, and report the averages.

We evaluate the proposed probabilistic SPM algorithms as follows. We first demonstrate the effectiveness of probabilistic pruning. Then, we test the scalability of our algorithms and report the CPU cost of each algorithm under various parameter settings. In addition to reporting the running times, we also monitor the memory usage of our algorithms, and report the peak memory used by each algorithm in terms of %age of the total system memory (RAM). Finally, we evaluate the effectiveness of probabilistic SPM framework in the presence of noise δ, and contrast the results obtained from data in the presence of noise to those obtained from data without any noise.

6.2 Probabilistic pruning

Given a sequence $s = \langle s_1, \ldots, s_q \rangle$ and a source i, recall that the objective in probabilistic pruning is to compute an upper bound on the probability that s is supported by source i, and that we compute the upper bounds for s in BFS and DFS differently. In BFS, we store $L_{i,1}$ along with their probabilities for source i and compute the upper bound as follows:

$$\Pr[s \preceq D_i^p] \leq \Pr[\langle s_1 \rangle \preceq D_i^p] * \ldots * \Pr[\langle s_q \rangle \preceq D_i^p]. \tag{11}$$

In DFS, we already have computed the value $\Pr[\langle s_1, \ldots, s_{q-1} \rangle \preceq D_i^p]$ and obtain the upper bound as below:

$$\Pr[s \preceq D_i^p] \leq \Pr[\langle s_1, \ldots, s_{q-1} \rangle \preceq D_i^p] * \Pr[\langle s_q \rangle \preceq D_i^p]. \tag{12}$$

To show the effectiveness of probabilistic pruning, we report the percentage of the infrequent candidate sequences that passed apriori pruning and were later eliminated by the probabilistic pruning, for candidate 2-sequences onwards. Note that the apriori pruning does not help for candidate 2-sequences, and probabilistic pruning is the only option.

We now describe our experiments. We choose `gazelle` and two representative synthetic datasets, namely `C10D10K` and `C20D10K` converted to SLU with $\delta = 10\,\%$, and report the results both for BFS and DFS in Fig. 1, for $\theta = 5$ and $10\,\%$ for synthetic datasets and for $\theta = 0.02$ and $0.04\,\%$ for `gazelle`. We have following observations:

1. We observe that probabilistic pruning is particularly effective in eliminating potential infrequent candidate 2-sequences both for synthetic and real datasets. We observe over

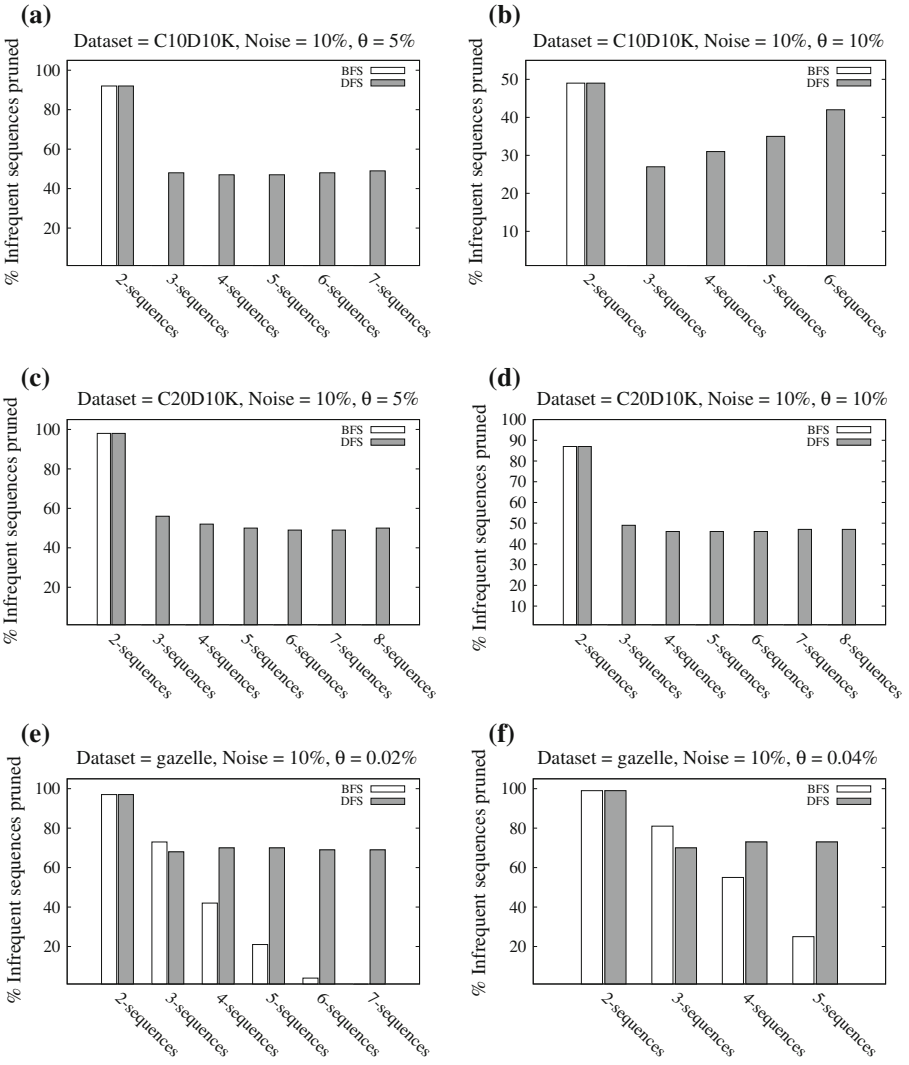


Fig. 1 Effectiveness of probabilistic pruning for BFS and DFS. In these graphs, each bar indicates the percentage of infrequent candidate sequences eliminated by probabilistic pruning that passed apriori pruning

90 % reduction by probabilistic pruning for infrequent candidate 2-sequences in most cases (all sub-figures of Fig. 1 except (b, d)).

2. We also observe that probabilistic pruning is more effective for relatively harder instances, for example, for `C10D10K` at $\theta = 5$ versus 10 %.
3. We observe that in BFS, probabilistic pruning does not help for synthetic datasets for candidate 3-sequences onwards, and becomes progressively less effective for `gazelle` as well for candidate 3-sequences onwards whereas in DFS, we see an overall reduction in the number of infrequent candidate sequences—upto a 50 % reduction in the infrequent candidate sequences for synthetic datasets and upto a 70 % reduction for `gazelle`—both for synthetic and real datasets. It might suggest that a tighter upper bound (Eqs. 11

vs. 12) needs to be computed for BFS as well, however, storing the probability with which a $(j-1)$ prefix of a j-sequence is supported by source i for all candidate j-sequences with every source, is memory intensive and would limit the execution of BFS even for smaller datasets.

We conclude from the above observations that probabilistic pruning is effective for candidate 2-sequences both for BFS and DFS. However, probabilistic pruning either did not help or was less effective in BFS for candidate 3-sequences onwards, whereas it was effective overall in DFS, both for synthetic and real datasets. We therefore, turn probabilistic pruning "ON" for BFS only for candidate 2-sequences, and turn probabilistic pruning "ON" for DFS for the entire duration of the algorithm.

6.3 Scalability analysis

We evaluate the probabilistic SPM algorithms, namely BFS, DFS and PGA, as follows. We first demonstrate the scalability of these algorithms, and report the CPU time of each algorithm under various parameter settings. In addition to reporting the CPU times, we also monitor the memory usage of these algorithms and report the peak memory used by each algorithm in terms of percentage of the total system memory (RAM). Finally, we contrast the performance of the probabilistic SPM algorithms with each other.

6.3.1 CPU cost

We consider four parameters in our experiments, the support threshold θ, average number of events per source sequence C, the number of source sequences D and the number of sources in the distribution W. Clearly, if the other three parameters are fixed, decreasing the θ values, or increasing: (a) the average number of events per source sequence C, (b) the number of source sequences D or (c) the number of sources in W, all make an instance harder. For IBM Quest datasets, we test our algorithms against four parameters, i.e. for varying θ values, for increasing C, for increasing D, and for increasing number of sources in W. As the number of source sequences D and the average number of events per source C is fixed for `gazelle`, we only test our algorithms for varying θ values in case of `gazelle`.

Varying θ. In the first set of experiments, Fig. 2a, b, we test our algorithms for varying θ values for `gazelle`, and for a representative synthetic dataset `C10D10K`. We observe that for both synthetic and real data sets, the running time increases quite rapidly as θ decreases (note that the y-axis is logarithmic).

To get an insight into this behaviour, we obtain the total number of frequent sequences for varying θ values in Fig. 3a, and also the distributions of frequent sequences in Fig. 4a, b, for the set of experiments reported in Fig. 2a, b. We observe that decreasing θ results in a sharp increase in the number of frequent sequences (Fig. 3a), and we can also see that the number of frequent sequences increase for sequences of all lengths (Fig. 4a, b) and consequently, we witness an increase in the running times for all algorithms.

Increasing C. In another set of experiments, Fig. 2c, d, we test the scalability of our algorithms for increasing average number of events per source sequence C, by keeping the number of source sequences D constant at 10K, and report the running times for two values of θ, for $\theta = 25$ and 12.5 %.

The running time graphs in Fig. 2c, d, show that a linear increase in C results in an exponential increase in the running times for all algorithms. Similar to our earlier experiments

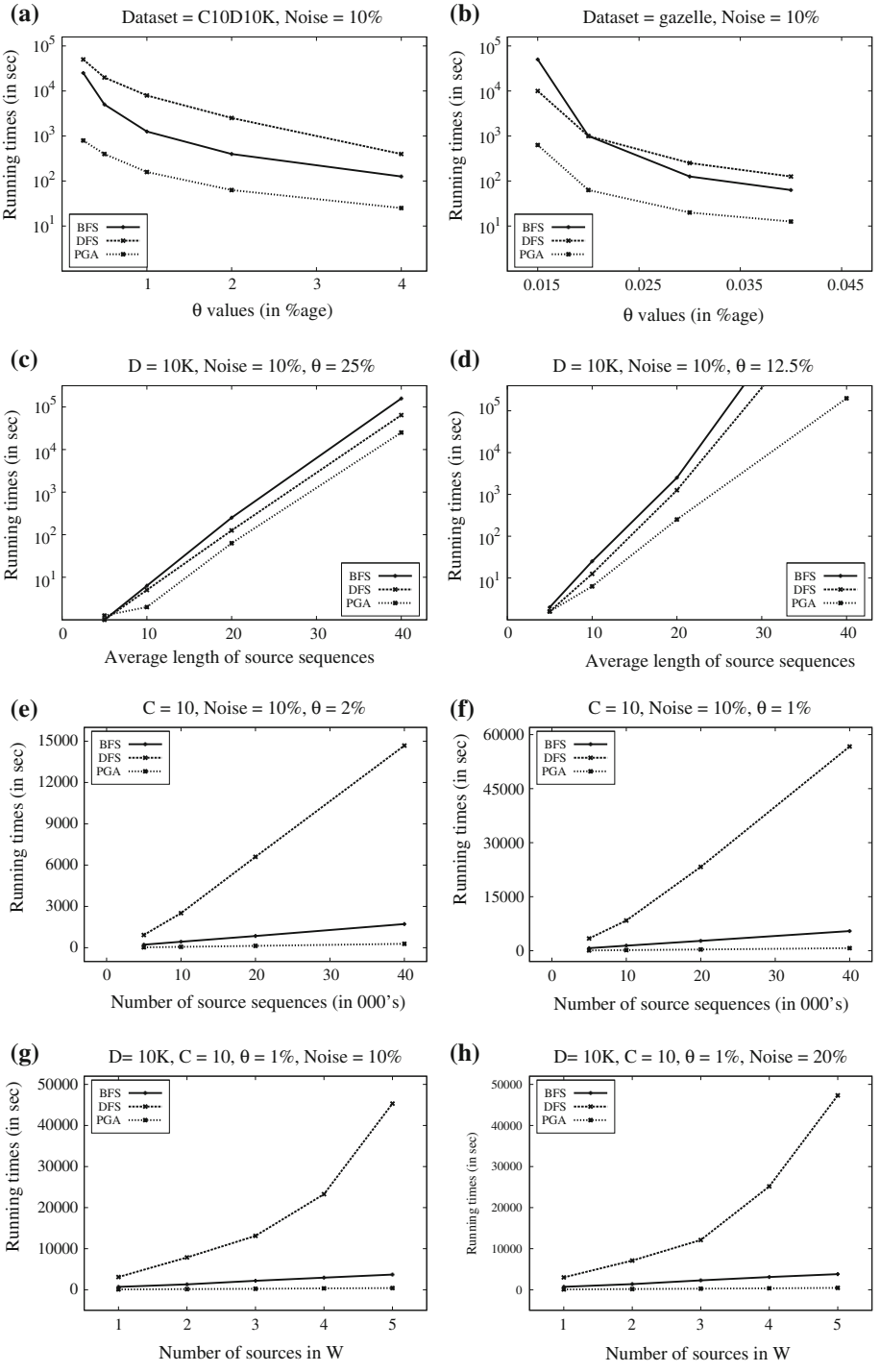


Fig. 2 CPU time (in seconds) for BFS, DFS and PGA under different parameter settings for `gazelle` and representative synthetic datasets

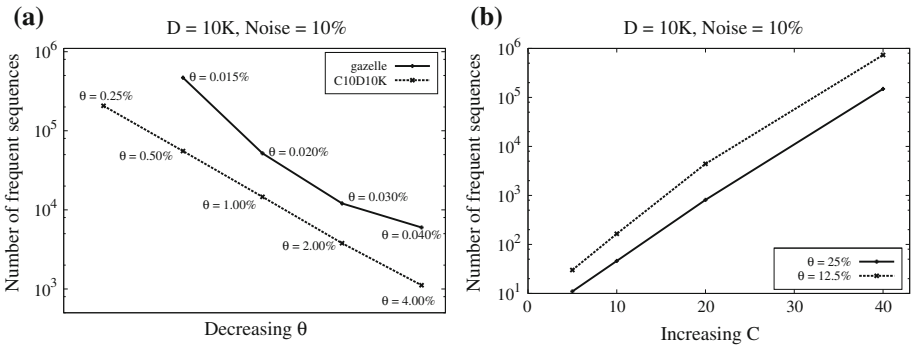


Fig. 3 Number of frequent sequences, for varying θ values for `gazelle` and `C10D10K`, and for increasing C (Fig. 2)

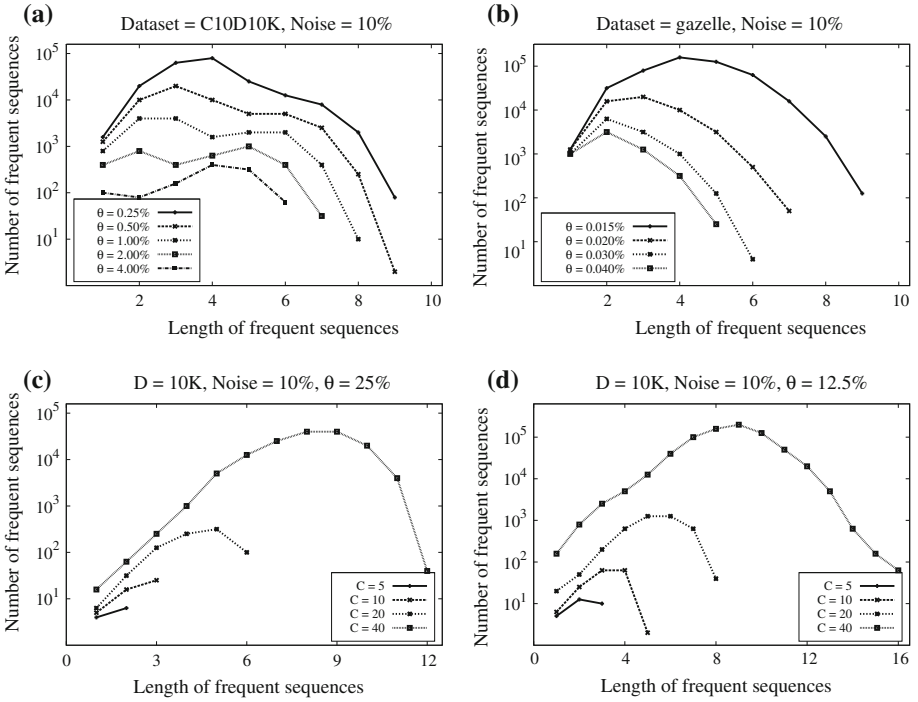


Fig. 4 Distribution of frequent sequences for `gazelle` and representative synthetic datasets for the set of experiments in Fig. 2a–d

for varying θ values, we obtain the total number of frequent sequences in Fig. 3b, and the distributions of frequent sequences in Fig. 4c, d, for the set of experiments in Fig. 2c, d. We observe that an increase in C results in rapid increase in the number of frequent sequences (Fig. 3b), and almost doubles the length of maximal frequent sequences (Fig. 4c, d), which would suggest the increased running itmes observed.

Increasing D. We also test the scalability of our algorithms for increasing number of source sequences D, Fig. 2e, f. We set the average number of events per source sequence $C = 10$, and report the running times for representative θ values at $\theta = 2$ and 1 %.

We observe that all our algorithms scale linearly for increasing D, although the running time graphs show that the rate of increase of the CPU cost for DFS is greater than for BFS. Note that the distributions of frequent sequences or lengths of maximal frequent sequences are largely unaffected as D increases, and we do not report those explicitly.

Increasing $|W|$. We test the scalability of our algorithms for increasing number of sources in W, Fig. 2g, h. We set the number of source sequences $D = 10$K, average number of events per source sequence $C = 10$, and report the running times for representative δ values at $\delta = 10$ and 20 %.

We observe that all our algorithms scale linearly for increasing $|W|$, although the running time graphs show that the rate of increase of the CPU cost for DFS is greater than for BFS and PGA, similar to the increasing D case.

Comparison of Algorithms. We now contrast our algorithms in terms of CPU time for the set of experiments in Fig. 2. We first focus on the candidate generation algorithms, BFS and DFS. We observe that whilst BFS is more efficient than DFS in CPU cost (Fig. 2a, e, f), DFS is better for increasing C (Fig. 2c, d) and also for `gazelle` at $\theta = 0.015$ %.

We can explain this behaviour considering the following key differences in BFS and DFS.

1. The candidate generation mechanism in DFS is different from BFS—DFS joins L_{j-1} with L_1 to obtain C_j, whereas BFS joins L_{j-1} with itself for the purpose—and thus, when the size of L_1 is larger than L_{j-1}, more candidate sequences are generated for DFS as compared to BFS. However, when L_{j-1} is significantly larger than L_1, for example for `gazelle` at $\theta = 0.015$ % or for `C40D10K` at $\theta = 25$ %, CPU cost for BFS is higher than DFS.
2. In DFS, only partial apriori pruning is possible, whereas BFS makes full use of the apriori pruning and thus in DFS, considerably more candidate sequences need to be considered for the later stages, i.e. probabilistic pruning and expected support computation.
3. Probabilistic pruning helps DFS more than the BFS as shown in Fig. 1.

Thus, depending on the interplay of these three factors, namely (1) number of candidate sequences generated (2) partial versus full apriori pruning and (3) effectiveness of probabilistic pruning, BFS may outperform DFS, or vice-versa.

Now comparing candidate generation algorithms with the pattern-growth algorithm, we can see that PGA is more efficient than both BFS and DFS in all our experiments (Fig. 2). In Fig. 2d at $C = 40$, we can see that only PGA manages to finish in a reasonable time, whereas both BFS and DFS do not finish even beyond 80 hours of execution.

Based on the above observations, we conclude that PGA is the most efficient overall in terms of CPU cost, whereas BFS and DFS have relatively higher CPU cost. We further conclude that while the performance of BFS and DFS depends on various parameter settings, PGA is rather oblivious to individual parameter settings and has a relatively stable behaviour.

6.3.2 Memory usage

We also monitor the memory usage of our algorithms for the set of experiments in Fig. 2, and report the peak memory used by each algorithm as a percentage of total system memory (RAM) in Fig. 5.

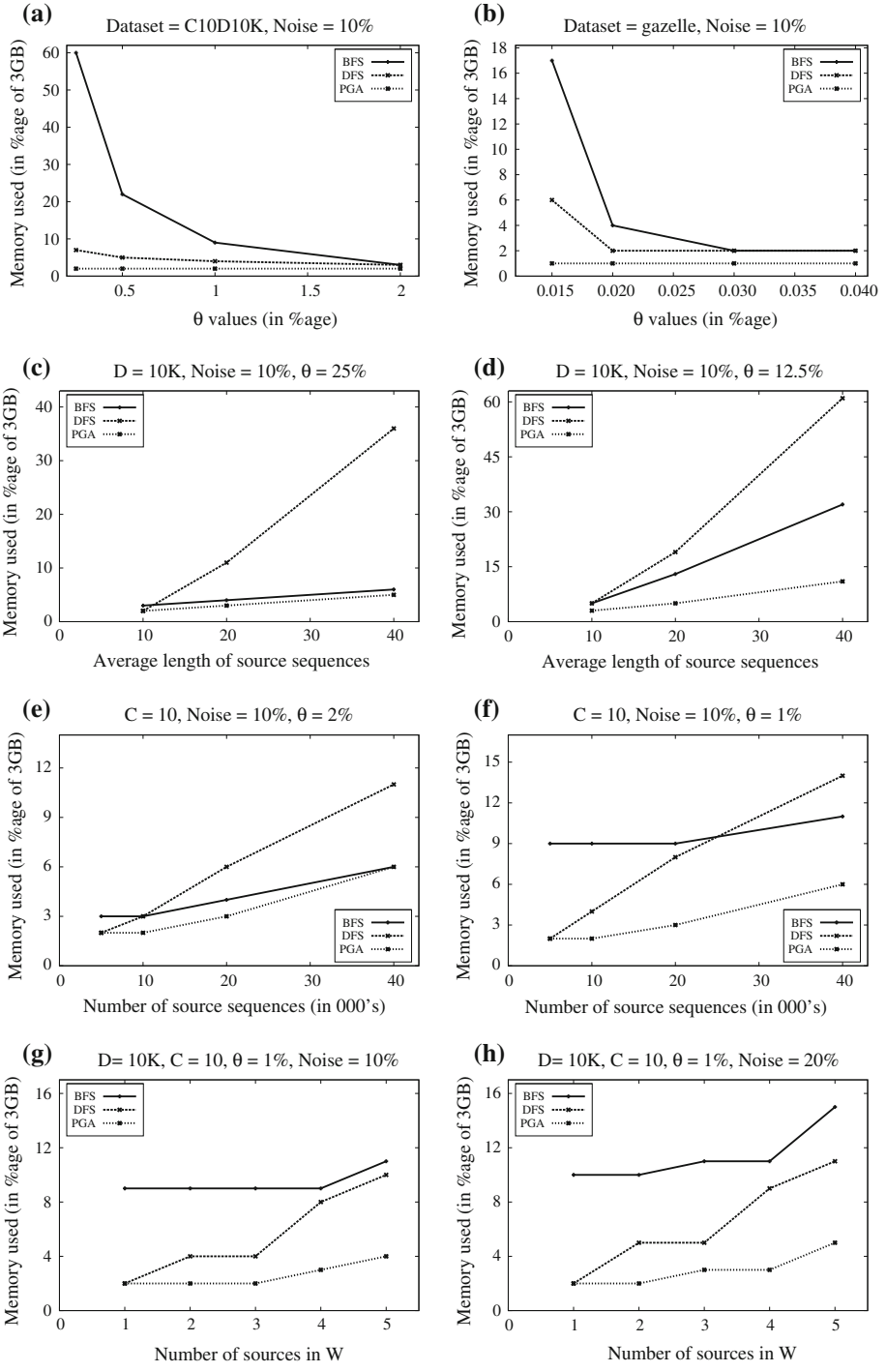


Fig. 5 Memory usage (in terms of %age of system memory used) for the set of experiments in Fig. 2

In Fig. 5a, b, we see that BFS has high memory requirements at very low θ values whereas DFS and PGA are relatively stable, for example at $\theta = 0.015\,\%$ for `gazelle` and at $\theta = 0.025\,\%$ for `C10D10K`, the memory usage for BFS increases sharply (upto 60 % of system memory), whereas it remains under 10 % for DFS and under 2 % for PGA.

In Fig. 5c, d, we observe that DFS has higher memory requirements for increasing C, whereas BFS consumes relatively less memory. We observe a sharp increase for BFS in Fig. 5d as compared to Fig. 5c at a low θ, for $\theta = 12.5$ versus 25 %. We observe that the memory usage for PGA remains rather stable and increasing C (Fig. 5c, d), or varying θ ($\theta = 25$ vs. 12.5 %) does not significantly affect the memory usage of PGA.

In Fig. 5e, f, we observe that all the algorithms require relatively less memory as compared to the other experiments in Fig. 5 (Fig. 5a–d). We observe that the memory usage for PGA almost doubles with a two-fold increase in the database size, for example for $D = 204$ versus 40 K in Fig. 5e, f. We also observe that the memory usage of PGA is not affected by decrease in θ, Fig. 5e, f at $\theta = 2$ versus 1 %, similar to the observations in Fig. 5a–d. In Fig. 5g, h, we see behaviours similar to Fig. 5e, f, as the memory requirements remain rather stable for increasing $|W|$.

As in BFS, all the candidate sequences in a phase are processed altogether, BFS has high memory requirements at low θ due to storing and processing all candidate sequences simultaneously. Further, we store some additional information in DFS and PGA to speed up the support computation. For instance, we store $B_{i,s}$ arrays along with the list of sources that support s in DFS, and only the $B_{i,s}$ arrays in PGA. It appears as if storing the $B_{i,s}$ arrays works well with PGA as it helps speed up the algorithm and the memory requirements also remain rather stable. Thus, we can conclude from the above set of experiments that PGA is the most scalable algorithm in terms of memory usage as well in contrast with both BFS and DFS.

We conclude from the set of experiments in this section that the pattern-growth approach extends the advantages that it is argued to have over candidate generation approaches in classical SPM setting, to the probabilistic case as well. In the candidate generation approaches, whilst BFS suffers from high cost of maintaining candidate sequences at low θ values, DFS can not take full advantage of the apriori pruning and therefore, performance of both the approaches is deteriorated for relatively harder instances. In summary, PGA scales well both in terms of CPU cost and memory usage as compared to BFS and DFS for the set of experiments we consider.

6.4 Effectiveness of the probabilistic SPM framework

We now report on an evaluation of effectiveness of the probabilistic SPM framework, specifically how well the probabilistic SPM approach is able to extract meaningful data in the presence of noise δ. Note that such studies have not been performed for probabilistic frequent itemset mining in the literature [10,11,13].

We discuss the methodology we use. As discussed previously, the datasets used in our experiments are all deterministic datasets, and we artificially introduce uncertainty (noise δ) into our datasets. An advantage of generating probabilistic data this way is that we can compare the frequent sequences obtained from deterministic data with those obtained from probabilistic data, to assess the effectiveness of probabilistic SPM framework in the presence of noise δ. We use the standard measures of *precision* and *recall* from information retrieval for the purpose [29, Chapter 8], which are defined as follows: given a set of frequent sequences R retrieved from a deterministic dataset, let R' be the set of frequent sequences retrieved

Table 8 A deterministic database (i) converted to SLU databases (ii–iv)

	Source sequence
(i)	
D_X	$(a_1)(a_2)$
D_Y	(a_3)
D_Z	–
	p-sequence
(ii)	
D_X^p	$(a_1 : 1.0)(a_2 : 1.0)$
D_Y^p	$(a_3 : 0.6)$
D_Z^p	$(a_3 : 0.4)$
(iii)	
D_X^p	$(a_1 : 1.0)(a_2 : 1.0)(a_3 : 0.4)$
D_Y^p	$(a_3 : 0.6)$
D_Z^p	–
(iv)	
D_X^p	$(a_1 : 0.5)(a_2 : 0.5)(a_3 : 0.5)$
D_Y^p	$(a_1 : 0.5)(a_3 : 0.5)$
D_Z^p	$(a_2 : 0.5)$

The events a_1, a_2 and a_3 represent the same itemset $\{a\}$ but with different event-ids. The support of the sequence $\langle(a)\rangle$ is 2 in the deterministic case, and the expected support of $\langle(a)\rangle$ is 2, 1.6 and $(0.875 + 0.75 + 0.5 =)$ 2.125, respectively, in SLU databases (ii–iv)

from the probabilistic dataset obtained by adding noise to the deterministic dataset. Then,

$$Precision = \frac{|R \cap R'|}{|R'|}, \; Recall = \frac{|R \cap R'|}{|R|}.$$

We evaluate the probabilistic SPM framework using both real and synthetic datasets. We report *overall* precision and recall for the complete set of frequent sequences, as well as for frequent k-sequences for $k = 1, \ldots, 7$ (i.e. precision and recall for sequences of individual length). In our experiments, we vary the noise parameter δ from 5 to 20 % both for real and synthetic datasets, and keep the rest of the parameters fixed. Observe that as a result of the noise introduced by us, the support of an individual sequence in the resulting probabilistic database may be equal to, less than or more than its support in the original deterministic dataset, as shown in Table 8.

6.4.1 Synthetic datasets

In the first set of experiments, Table 9, we consider the dataset `C10D10K` at $\theta = 2$ and 1 %, and report the precision and recall results for different values of δ, for $\delta = 5$, 10 and 20 %. In another set of experiments, Table 10, we consider the dataset `C20D10K` at $\theta = 20$ and 10 %, and report the precision and recall results similar to Table 9.

We observe that for the datasets we consider, namely `C10D10K` (for $\theta = 2$, 1 %) and `C20D10K` (for $\theta = 20$, 10 %), change in C or θ, does not significantly affect precision/recall. However, when δ is varied (between 5 to 20 % in our experiments), both precision and recall are affected. Thus, whilst precision is slightly worse than recall when δ is increased, we get

Table 9 Precision and recall results for synthetic dataset C10D10K

Dataset	*k*-sequences							*overall*
C10D10K	1	2	3	4	5	6	7	
$\theta = 2\,\%$								
$\delta = 5\,\%$								
Precision	1.00	0.95	0.96	0.99	0.96	0.98	1.00	0.97
Recall	1.00	0.98	0.97	0.98	0.97	0.96	0.95	0.98
$\delta = 10\,\%$								
Precision	1.00	0.92	0.92	0.98	0.90	0.95	1.00	0.94
Recall	1.00	0.98	0.97	0.97	0.96	0.93	0.77	0.97
$\delta = 20\,\%$								
Precision	1.00	0.86	0.81	0.89	0.82	0.84	1.00	0.86
Recall	1.00	0.96	0.94	0.95	0.94	0.93	0.34	0.95
$\theta = 1\,\%$								
$\delta = 5\,\%$								
Precision	1.00	0.95	0.93	0.97	0.96	0.95	0.99	0.95
Recall	1.00	0.99	0.97	0.97	0.98	0.98	0.88	0.97
$\delta = 10\,\%$								
Precision	1.00	0.93	0.88	0.95	0.91	0.90	0.98	0.92
Recall	1.00	0.98	0.95	0.95	0.95	0.96	0.84	0.96
$\delta = 20\,\%$								
Precision	1.00	0.89	0.80	0.87	0.84	0.82	0.96	0.86
Recall	1.00	0.96	0.92	0.90	0.92	0.94	0.76	0.93

over 90 % overall recall in our experiments in Tables 9 and 10. Further, we get over 80 % overall precision as well except for C20D10K for $\theta = 20\,\%$ and $\delta = 20\,\%$, where it is 77 % (Table 10).

We next give precision and recall against frequent *k*-sequences of length upto 7. We have the following observations:

1. We observe that recall is near perfect for small values of k and declines when the value of k increases, e.g. for $k = 6$ or 7, whereas precision is affected by an increase in δ rather than an increase in the sequence length.
2. We also observe that recall is not good for long sequences and especially, when δ is also high, for example, for 7-sequences at $\delta = 20\,\%$, in C10D10K at $\theta = 2\,\%$ or in C20D10K at $\theta = 20\,\%$. This is not entirely unexpected as longer sequences would tend to have lower support (and would thus be closer to the threshold) in the deterministic database, and the noise would tend to further lower the support of long sequences.
3. It is also interesting to note that whilst overall recall remains relatively unaffected for different values of θ, we get slightly better recall for lower values of θ, for instance, for 7-sequences in C20D10K at $\theta = 20\,\%$ versus $\theta = 10\,\%$.

We conclude from the set of experiments in Tables 9 and 10 that for the datasets we consider and the expected support and noise thresholds, we get encouraging precision/recall results overall as well as for sequences of different lengths, even when noise is relatively high.

Table 10 Precision and recall results for synthetic dataset C20D10K

Dataset	*k*-sequences							*overall*
C20D10K	1	2	3	4	5	6	7	
$\theta = 20\,\%$								
$\delta = 5\,\%$								
Precision	1.00	1.00	0.96	0.92	0.93	0.89	1.00	0.92
Recall	1.00	1.00	1.00	0.99	0.99	1.00	0.59	0.99
$\delta = 10\,\%$								
Precision	1.00	1.00	0.94	0.83	0.84	0.85	1.00	0.86
Recall	1.00	1.00	1.00	0.99	1.00	1.00	0.44	0.98
$\delta = 20\,\%$								
Precision	1.00	1.00	0.87	0.79	0.71	0.79	1.00	0.77
Recall	1.00	1.00	1.00	0.97	0.99	1.00	0.31	0.97
$\theta = 10\,\%$								
$\delta = 5\,\%$								
Precision	1.00	0.93	0.96	0.94	0.94	0.95	0.99	0.96
Recall	1.00	0.99	1.00	1.00	0.98	0.99	0.98	0.99
$\delta = 10\,\%$								
Precision	1.00	0.91	0.96	0.92	0.86	0.88	0.96	0.91
Recall	1.00	1.00	0.98	1.00	0.97	0.99	0.97	0.98
$\delta = 20\,\%$								
Precision	0.98	0.88	0.91	0.85	0.81	0.78	0.86	0.83
Recall	1.00	0.99	0.95	0.96	0.96	0.96	0.96	0.95

6.4.2 gazelle

We now evaluate the effectiveness of probabilistic SPM framework for gazelle. In our experiments (Table 11), we report precision and recall results for two values of θ, for $\theta = 0.04$ and 0.03 % and vary the noise parameter δ between 5 to 20 % similar to synthetic datasets.

We observe that although precision is generally good, recall is rather poor. Further, the longest sequences seem to be worse affected by noise. For example, whilst the overall precision is near perfect for the set of experiments in Table 11, even the highest overall recall is only 0.67 which is for $\theta = 0.04\,\%$ at $\delta = 5\,\%$, whereas overall recall gets as low as 20 % for $\theta = 0.03\,\%$ and for $\delta = 20\,\%$. A similar observation can be made about sequences of individual lengths, i.e. precision is perfect or near perfect but recall is very low. In other words, we can say that although most of the extracted sequences are relevant, there are not many of them.

Our understanding of a low recall is that the sequences with the highest expected support are not able to pass the support threshold θ or in other words, θ is too high. It might suggest that the θ values need to be fine tuned, or in other words, revised downwards in order to improve recall (see [30] for a discussion on soft thresholds for pattern mining). It is obvious that revising θ downwards in order to improve recall will be at the cost of some precision, i.e. precision is likely to deteriorate as a consequence. One important issue is that it is not clear how to systematically adjust the θ values. In our experiments, we revise θ by setting $\theta = \theta * (1 - \delta)$ (there could be other ways as well), and the rows labelled as "Change" in

Table 11 Precision and recall results for `gazelle`

Dataset	*k*-sequences							*overall*
`gazelle`	1	2	3	4	5	6	7	
$\theta = 0.04\,\%$								
$\delta = 5\,\%$								
Precision	1.00	1.00	1.00	1.00	1.00	1.00	0.00	*1.00*
Recall	1.00	0.81	0.57	0.40	0.20	0.08	0.00	*0.67*
$\delta = 10\,\%$								
Precision	1.00	0.99	1.00	1.00	1.00	0.00	0.00	*1.00*
Recall	1.00	0.68	0.39	0.20	0.06	0.00	0.00	*0.53*
$\delta = 20\,\%$								
Precision	0.99	0.98	1.00	1.00	0.00	0.00	0.00	*1.00*
Recall	1.00	0.46	0.16	0.03	0.00	0.00	0.00	*0.41*
$\theta = 0.03\,\%$								
$\delta = 5\,\%$								
Precision	1.00	1.00	1.00	1.00	1.00	1.00	1.00	*1.00*
Recall	1.00	0.77	0.50	0.31	0.17	0.05	0.01	*0.50*
$\delta = 10\,\%$								
Precision	1.00	0.99	1.00	1.00	1.00	1.00	0.00	*1.00*
Recall	1.00	0.65	0.31	0.14	0.04	0.00	0.00	*0.36*
$\delta = 20\,\%$								
Precision	1.00	0.98	1.00	1.00	1.00	0.00	0.00	*0.99*
Recall	1.00	0.45	0.11	0.02	0.01	0.00	0.00	*0.20*

Table 12 show the effect of this adjustment on precision and recall. For example, a value +0.20 in the "Change" row under recall means that recall improved by 20 % as a result of revising θ, whereas the values in the corresponding precision and recall rows show the updated precision and recall values.

We report the updated precision and recall results after revising θ in Table 12. As expected, we see some improvement in recall at the cost of precision. For example, observe an improvement of over 40 % in overall recall, at the cost of a nearly 20 % decline in precision, for $\theta = 0.03\,\%$ and $\delta = 20\,\%$. The results are encouraging for sequences of individual lengths as well; for example, observe 54 and 59 % improvement in recall for frequent 2- and 3-sequences, respectively, for $\theta = 0.03\,\%$ and $\delta = 20\,\%$. However, precision declines as a consequence: observe a 31 and 5 % decline in precision, respectively, in the aforementioned case. Further, the recall is still not good for longer sequences, e.g. for 6- or 7-sequences and this might suggest that the θ needs to be fine tuned in each phase or alternatively, for sequences of each length. However, it is not clear how to do this.

Thus, we can suggest from the set of experiments we consider that θ needs to be fine tuned in order to get better precision/recall results for `gazelle`.

Concluding our discussion on the effectiveness of the probabilistic SPM framework in the presence of noise, we say that we get encouraging precision/recall results overall as well as for sequences of individual lengths, for our considered synthetic datasets. However, the recall was rather poor for `gazelle` which we were able to improve considerably by an adjustment in θ, albeit at the cost of some precision.

Table 12 The updated precision and recall results for `gazelle` after θ is revised

Dataset	*k*-sequences							*Overall*
`gazelle`	1	2	3	4	5	6	7	
$\theta = 0.04\,\%$								
$\delta = 5\,\%$								
Precision	0.96	0.98	1.00	1.00	1.00	1.00	0.00	0.99
Change	−0.04	−0.02	0.00	0.00	0.00	0.00	0.00	−0.01
Recall	1.00	0.99	0.82	0.65	0.37	0.16	0.00	0.86
Change	0.00	+0.18	+0.25	+0.25	+0.17	+0.08	0.00	+0.19
$\delta = 10\,\%$								
Precision	0.94	0.88	0.99	1.00	1.00	1.00	0.00	0.93
Change	−0.06	−0.11	−0.01	0.00	0.00	0.00	0.00	−0.07
Recall	1.00	0.99	0.80	0.51	0.20	0.03	0.00	0.84
Change	0.00	+0.31	+0.41	+0.31	+0.14	+0.03	0.00	+0.31
$\delta = 20\,\%$								
Precision	0.84	0.67	0.95	1.00	1.00	0.00	0.00	0.78
Change	−0.15	−0.31	−0.05	0.00	0.00	0.00	0.00	−0.22
Recall	1.00	1.00	0.75	0.28	0.06	0.00	0.00	0.86
Revised θ	0.00	+0.54	+0.59	+0.25	+0.06	0.00	0.00	+0.45
$\theta = 0.03\,\%$								
$\delta = 5\,\%$								
Precision	0.96	0.99	1.00	1.00	1.00	1.00	1.00	1.00
Change	−0.04	−0.01	0.00	0.00	0.00	0.00	0.00	0.00
Recall	1.00	0.98	0.77	0.55	0.37	0.17	0.07	0.77
Change	0.00	+0.21	+0.27	+0.24	+0.20	0.12	0.06	+0.22
$\delta = 10\,\%$								
Precision	0.94	0.90	0.99	1.00	1.00	1.00	0.00	0.95
Change	−0.06	−0.09	−0.01	0.00	0.00	0.00	0.00	−0.05
Recall	1.00	0.98	0.75	0.44	0.20	0.03	0.00	0.68
Change	0.00	+0.33	+0.44	+0.30	+0.16	0.03	0.00	+0.32
$\delta = 20\,\%$								
Precision	0.89	0.69	0.91	0.99	1.00	0.00	0.00	0.80
Change	−0.11	−0.29	−0.09	−0.01	0.00	1.00	0.00	−0.19
Recall	1.00	1.00	0.72	0.25	0.05	0.00	0.00	0.61
Change	0.00	+0.55	+0.61	+0.23	+0.04	0.00	0.00	+0.41

The rows labelled as "Change" show the improvement/decline (+/-) in the precision and recall results after θ is revised

7 Conclusions and future work

We have considered the problem of finding all frequent sequences in an SLU database under the expected support measure. We have proposed three algorithms, two based on the candidate generation and one based on the pattern-growth framework; our empirical evaluation shows that the pattern-growth approach is more scalable than the candidate generation approaches, as observed in the deterministic case. In contrast, it has been noted that when mining frequent

itemsets from uncertain data, the pattern-growth approach (FP-tree) does not scale well [22]. We have also shown that the probabilistic SPM framework is effective in extracting meaningful patterns from SLU data using the expected support measure.

This is one of the first studies 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 an SLU database—the probabilistic frequentness [11] is provably intractable for SLU databases [21]. 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).

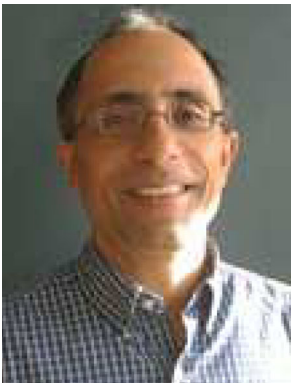
Acknowledgments We are grateful to the anonymous reviewers for making useful suggestions in improving this work.

References

1. Agrawal R, Srikant R (1995) Mining sequential patterns. In: Yu Philip S, Chen Arbee LP (eds) ICDE. IEEE Computer Society, pp 3–14. ISBN 0-8186-6910-1
2. Srikant R, Agrawal R (1996) Mining sequential patterns: generalizations and performance improvements. In: Apers Peter MG, Bouzeghoub Mokrane, Gardarin Georges (eds) EDBT, volume 1057 of LNCS. Springer, pp 3–17. ISBN 3-540-61057-X
3. Zaki MJ (2001) SPADE: An efficient algorithm for mining frequent sequences. Mach Learn 42(1/2):31–60
4. Pei J, Han J, Mortazavi-Asl B, Wang J, Pinto H, Chen Q, Dayal U, Hsu M (2004) Mining sequential patterns by pattern-growth: the PrefixSpan approach. IEEE Trans Knowl Data Eng 16(11):1424–1440
5. Ayres J, Flannick J, Gehrke Js, Yiu T (2002) Sequential pattern mining using a bitmap representation. In: KDD, pp 429–435. ACM. ISBN 1-58113-567-X
6. Aggarwal CC (ed) (2009) Managing and mining uncertain data. Springer, Berlin
7. Suciu D, Dalvi Nilesh N (2005) Foundations of probabilistic answers to queries. In: Özcan Fatma (ed) SIGMOD Conference. ACM, pp 963. ISBN 1-59593-060-4
8. Zhang Q, Li F, Yi K. Finding frequent items in probabilistic data. In: Wang [31], pp 819–832. ISBN 978-1-60558-102-6
9. Cormode G, Li F, Yi K (2009) Semantics of ranking queries for probabilistic data and expected ranks. In: ICDE. IEEE, pp 305–316. ISBN 978-0-7695-3545-6
10. Aggarwal CC, Li Y, Wang J, Wang J. Frequent pattern mining with uncertain data. In: Elder et al. [32], pp 29–38. ISBN 978-1-60558-495-9
11. Bernecker T, Kriegel H-P, Renz M, Verhein F, Züfle A. Probabilistic frequent itemset mining in uncertain databases. In: Elder et al. [32], pp 119–128. ISBN 978-1-60558-495-9
12. Chui CK, Kao B (2008) A decremental approach for mining frequent itemsets from uncertain data. In: Washio T, Suzuki E, Ting KM, Inokuchi A (eds) PAKDD, volume 5012 of LNCS. Springer, pp 64–75. ISBN 978-3-540-68124-3
13. Chui CK, Kao B, Hung E (2007) Mining frequent itemsets from uncertain data. In: Zhou Z-H, Li H, Yang Q (eds) PAKDD, volume 4426 of LNCS. Springer, pp 47–58. ISBN 978-3-540-71700-3
14. Sun L, Cheng R, Cheung DW, Cheng J (2010) Mining uncertain data with probabilistic guarantees. In: Rao B, Krishnapuram B, Tomkins A, Yang Q (eds) KDD. ACM, pp 273–282. ISBN 978-1-4503-0055-1
15. Wang L, Cheng R, Lee SD, Cheung David Wai-Lok (2010) Accelerating probabilistic frequent itemset mining: a model-based approach. In: Huang J, Koudas N, Jones GJF, Wu X, Collins-Thompson K, An A (eds) CIKM. ACM, pp 429–438. ISBN 978-1-4503-0099-5
16. Calders T, Garboni C, Goethals B (2010) Approximation of frequentness probability of itemsets in uncertain data. In: Webb GI, Liu B, Zhang C, Gunopulos D, Wu X (eds) ICDM. IEEE Computer Society, pp 749–754
17. Yang J, Wang W, Yu PS, Han J (2002) Mining long sequential patterns in a noisy environment. In: Franklin MJ, Moon B, Ailamaki A (eds) SIGMOD Conference. ACM, pp 406–417. ISBN 1-58113-497-5

18. Sun X, Orlowska ME, Li X (2003) Introducing uncertainty into pattern discovery in temporal event sequences. In: ICDM. IEEE Computer Society, pp 299–306. ISBN 0-7695-1978-4
19. Wikipedia. http://en.wikipedia.org/wiki/anpr—Wikipedia, the free encyclopedia, 2010. URL http://en.wikipedia.org/wiki/ANPR. [Online; accessed 30-April-2012]
20. Keilthy L (2008) ANPR system performance. In: Ruh M, Trost-Heutmekers G (eds) Parking trend international. European Parking Association, June 2008. http://www.parkingandtraffic.co.uk/Measuring Online; accessed 30-June-2012
21. Muzammal M, Raman R (2010) On probabilistic models for uncertain sequential pattern mining. In: Cao L, Feng Y, Zhong J (eds) ADMA (1), volume 6440 of LNCS, pp 60–72. Springer. ISBN 978-3-642-17315-8
22. Tong Y, Chen L, Cheng Y, Yu PS (2012) Mining frequent itemsets over uncertain databases. PVLDB 5(11):1650–1661
23. Hua M, Pei J, Zhang W, Lin X. Ranking queries on uncertain data: a probabilistic threshold approach. In: Wang [31], pp 673–686. ISBN 978-1-60558-102-6
24. Hooshadat M, Bayat S, Naeimi P, Mirian MS, Zaiane OR (2012) Uapriori: an algorithm for finding sequential patterns in probabilistic data. In: Kahraman C, Bozbura FT, Kerre EE (eds) UMKEDM. World Scientific, pp 907–912. ISBN 978-9814417730
25. Zhao Z, Yan D, Ng W (2012) Mining probabilistically frequent sequential patterns in uncertain databases. In: Rundensteiner EA, Markl V, Manolescu I, Amer-Yahia S, Naumann F, Ari I (eds) EDBT. ACM, pp 74–85. ISBN 978-1-4503-0790-1
26. Wan L, Chen L, Zhang C (2013) Mining frequent serial episodes over uncertain sequence data. In: Guerrini G, Paton NW (eds) EDBT, pp 215–226. ACM. ISBN 978-1-4503-1597-5
27. Achar A, Ibrahim A, Sastry PS (2013) Pattern-growth based frequent serial episode discovery. Data Knowl Eng 87:91–108
28. Kohavi R, Brodley C, Frasca B, Mason L, Zheng Z (2000) KDD-Cup 2000 organizers' report: peeling the onion. SIGKDD Explor 2(2):86–98
29. Manning CD, Prabhakar R, Hinrich S (2008) Introduction to information retrieval. Cambridge University Press, New York, NY, USA. ISBN 0521865719, 9780521865715
30. Ugarte W, Boizumault P, Loudni S, Crémilleux B (2012) Soft threshold constraints for pattern mining. In: Ganascia J-G, Lenca P, Petit J-M (eds) Discovery Science, volume 7569 of Lecture notes in computer science. Springer, pp 313–327. ISBN 978-3-642-33491-7
31. Wang JTL (ed) (2008) Proceedings of the ACM SIGMOD international conference on management of data, SIGMOD 2008, Vancouver, BC, Canada, June 10–12, ACM. ISBN 978-1-60558-102-6
32. Elder JF, Fogelman-Soulié F, Flach PA, Zaki MJ (eds) (2009) Proceedings of the 15th ACM SIGKDD international conference on knowledge discovery and data mining, Paris, France, June 28 - July 1, 2009. ACM. ISBN 978-1-60558-495-9

Muhammad Muzammal is an Assistant Professor in the Computer Science Department at Bahria University, Islamabad. He received Ph.D. degree in Computer Science from University of Leicester in 2012. His research interests are in data mining, uncertain data and human-centred computing.



Rajeev Raman obtained his B. Tech and Ph.D. degrees from IIT Delhi and the University of Rochester in 1986 and 1993 respectively. He held research positions at the Max Planck Institute for Informatics and at the University of Maryland before joining King's College London as a Lecturer (Assistant Professor). He has been a Professor at the University of Leicester since 2001 (and served as Head of the Computer Science Department from 2003–2006). His primary research interests are in algorithms and complexity, particularly data structures and parallel algorithms, and applications of randomised techniques. Recently his interests have lain in *succinct*, or highly space-efficient, data structures, and is currently looking to apply succinct data structures to improve the scalability of data mining algorithms. He is on the editorial boards of the *J. Discrete Algorithms* (Elsevier) and the *Computer Journal* (OUP) and has served as PC member or chair of about 40 conferences.