Efficient Matching of Substrings in Uncertain Sequences *

Yuxuan Li† James Bailey‡ Lars Kulik§ Jian Pei¶

Abstract

Substring matching is fundamental to data mining methods for sequential data. It involves checking the existence of a short subsequence within a longer sequence, ensuring no gaps within a match. Whilst a large amount of existing work has focused on substring matching and mining techniques for certain sequences, there are only a few results for uncertain sequences. Uncertain sequences provide powerful representations for modelling sequence behavioural characteristics in emerging domains, such as bioinformatics, sensor streams and trajectory analysis. In this paper, we focus on the core problem of computing substring matching probability in uncertain sequences and propose an efficient dynamic programming algorithm for this task. We demonstrate our approach is both competitive theoretically, as well as effective and scalable experimentally. Our results contribute towards a foundation for adapting classic sequence mining methods to deal with uncertain data.

1 Introduction

Substring matching is a fundamental problem in pattern mining and plays a key role in many important data mining tasks for sequential pattern mining [4, 20, 26, 5, 19] and top-k querying [25, 14, 13]. A substring corresponds to a subsequence of a longer sequence *without gaps between adjacent items in the match* (e.g. *AB* is a substring of *ACDAB*, but *DB* is not a substring of *ACDAB*).

Substring matching is a core operation useful in a range of applications. For example, i) searching for a short substring within a very long sequence, such as a word within a document, or ii) checking the frequency of a substring in a database of sequences, as a prelude to enumerating frequent subsequences, or for ranking the sequences in a top-k best fashion. Applications areas that rely on substring matching include bioinformatics (DNA and protein search), knowledge discovery for moving object database trajectories, web log analysis and text mining.

Whilst a large amount of existing work has focused on substring matching and mining techniques for certain sequences (i.e. where there is no ambiguity about the elements in each sequence), the landscape of results for uncertain sequences has been less explored. Uncertain sequences arise naturally in applications where i) the data measurements are imprecise, incomplete or unreliable, such as streams of sensor measurements, RFID measurements and trajectory measurements, or ii) flexible descriptions of sequence properties are required, such as binding profiles of proteins in DNA sequences, or iii) observations are private and thus sequences of observations may have artificial noise or uncertainty deliberately introduced.

$\hat{s}$	$\hat{s}$ [1]	$\hat{s}$ [2]	$\hat{s}$ [3]	$\hat{s}$ [4]	$\hat{s}$ [5]	$\hat{s}$ [6]	$\hat{s}$ [7]	$\hat{s}$ [8]	$\hat{s}$ [9]	$\hat{s}$ [10]
A	0.6	0	0.1	0.1	0.2	0.2	0.3	0.3	0.2	0.2
C	0.2	0	0.6	0.3	0.4	0.1	0.4	0.2	0.4	0.2
G	0.1	1.0	0.2	0.1	0.1	0.3	0.1	0.1	0.2	0.1
T	0.1	0	0.1	0.5	0.3	0.4	0.2	0.4	0.2	0.5

Figure 1: An uncertain sequence of length 10 using alphabet {A,C,G,T}.

Constructing representations of and mining techniques for uncertain data has recently attracted much interest [10, 13] (see [2] for a survey). However, with a few exceptions, much of this work has focused on uncertainty for vector/itemset data, rather than on uncertainty for sequence data.

In this paper, we tackle the problem of substring matching for uncertain sequences. Our main contribution is the development of an efficient algorithm to compute the following substring matching query: *Given a certain sequence q and an uncertain sequence $\hat{s}$, what is the probability that q is a substring of $\hat{s}$?* Figure 1 shows an example of an uncertain sequence $\hat{s}$ using alphabet $\{A,C,G,T\}$. A sequence is called uncertain if the occurrence of a character (from a given alphabet) at each position of that sequence is probabilistic. An example query is **what is the probability that the (certain) sequence *AGCTCT* is a substring of $\hat{s}$?** In other words, what is the probability that *AGCTCT* occurs in $\hat{s}$ with no gaps between any two adjacent characters of the uncertain sequence? (In Section 2 we will provide a formal description of uncertainty and the substring matching task).

The outcome of the substring matching query can be used as a core component within other data mining tasks, which we will describe in this paper: (1) computing the frequentness probability for a substring, (2) mining probabilistic frequent substring patterns, and (3) computing the similarity between two uncertain sequences.

*This research is supported under the Australian Research Council's Discovery Projects funding schema (project number DP110100757).

†University of Melbourne, Australia. Email: yuxuan.li@unimelb.edu.au

‡University of Melbourne, Australia. Email: baileyj@unimelb.edu.au

§University of Melbourne, Australia. Email: lkulik@unimelb.edu.au

¶Simon Fraser University, Canada. Email: jpei@cs.sfu.ca

1.1 Challenges and Contributions Matching a (certain) substring query pattern in an uncertain sequence is a challenging problem for several reasons. (1) As the length of an uncertain sequence increases, the number of possible worlds grows exponentially. This makes a naive technique that exhaustively enumerates all possible worlds infeasible. (2) Substring matching requires that no gaps are allowed within the match. This imposes a constraint on how the matching may be performed and in the case of an uncertain sequence, effectively rules out the use of techniques which can scan through a sequence in a markov or memoryless fashion. Our experience is that developing an algorithm for substring matching for uncertain sequences is considerably more complex than subsequence matching for uncertain sequences (where unlimited gaps are allowed). (3) The development of an algorithm requires partitioning the problem into a series of independent, smaller problems. Care is needed to avoid double counting possible worlds across these smaller problems.

In this paper, we address these challenges. In particular, we make the following contributions:

- We develop a dynamic programming approach to calculate the substring matching probability which has linear time (assuming a constant query size) complexity. This is the first such linear time result we are aware of for matching strings in uncertain sequences with an arbitrary alphabet .
- We illustrate applications of substring matching for three important scenarios: (1) computing the frequentness probability for a substring in uncertain sequence databases. (2) mining probabilistic frequent substrings from uncertain sequence databases. (3) computing the similarity between two uncertain sequences.

1.2 Related Work The problem of classical sequential pattern mining has been an area of extensive research in the context of certain databases [4, 20, 26, 5, 19]. The general task of sequence similarity calculation was introduced in [3] and also studied in [17, 11].

In the context of uncertain databases, the problem of uncertain frequent itemset mining in probabilistic databases was earlier studied under the expected support measure in [1, 10, 9, 8]. However, work in [27, 6, 21] found that the use of expected support may lead to the loss of important patterns. Thus, the use of a probabilistic frequentness measure has recently become more popular, based on *possible world semantics*. The characteristics of a pattern are assessed by its confidence. For example, a pattern is called frequent in [6, 16] if its probabilistic measurement exceeds a confidence threshold τ. We will use possible world semantics to calculate the substring matching probability (c.f. Section 2). A recent survey comparing expected support and frequentness probability is given in [22].

For the problem of uncertain sequential patterns, there is less work in the literature [18, 28, 16, 23]. In [18], the authors measure the frequentness of a pattern in uncertain event databases based on the expectation measurement, but this may sometimes lead to the loss of interesting patterns [6, 21]. Work by Zhao et al. [28] addressed uncertain sequence mining under two different models of uncertainty, sequence level uncertainty and element level uncertainty. Their study is complementary, yet distinct from our work in this paper, since they focus on the case where unlimited gaps are permitted when matching a query subsequence against a longer sequence. Likewise, work by Wan et al. [23] examines the mining of frequent serial episodes within an uncertain sequence. Here again, the problem studied allows unlimited gaps in the matching, whereas our work enforces a constraint that disallows gaps. In a more recent study by Li et al. [16], the probabilistic spatial-temporal pattern with gap constraints was introduced. Gaps are allowed in a spatio-temporal pattern provided they satisfy a maximum gap constraint. The problem studied corresponds to a specialised type of matching using an alphabet with only a single character that may either occur or not occur (in contrast to this paper where an alphabet of unlimited size is permitted, allowing more rich applications to be modelled). A study in [12] examined the approximation between two uncertain strings based on their edit distance (insertions, deletions or substitutions), which is very different from our problem definition. An approach used in [24] is only able to output the approximate result of the substring matching probability. In comparison, we propose an efficient exact algorithm in this paper.

2 Preliminaries and Problem Definition

Let $s = \{s[1], s[2], ..., s[n]\}$ be a sequence that contains n characters chosen from a finite alphabet Λ. Let $q = \{q[1], q[2], ..., q[m]\}$ be a sequence that contains m ($m \leq n$) characters chosen from the same alphabet Λ. We begin by first defining substring containment, which essentially corresponds to matching a query subsequence in a longer sequence, without allowing any gaps in the matching.

DEFINITION 2.1. *A sequence q is called a **substring** of another sequence s iff there exists a k ($0 \leq k \leq n-m$), such that $\forall i \in [1,m] : q[i] = s[i+k]$. We use the notation $q \sqsubseteq s$ to denote that q is a substring of s.*

EXAMPLE 1. *$q = AC$ is a substring of $s = GACT$. $q = AT$ is not a substring of $s = GACT$. $q = ACT$ is a substring of $GACTGACT$.*

Observe that the index k in Definition 2.1 may not be unique and the substring q may in fact match in the sequence s in several places. The key point is that there needs to exist at least one match for q, in order for it to qualify as a substring.

$\hat{s}$	$\hat{s}$ [1]	$\hat{s}$ [2]	$\hat{s}$ [3]	$\hat{s}$ [4]
A	0.1	0	0.2	0
C	0.2	1.0	0.4	1.0
G	0.3	0	0.2	0
T	0.4	0	0.2	0

k	World	$P(W_k)$
1	ACAC	0.02
2	ACCC	0.04
3	ACGC	0.02
4	ACTC	0.02
5	CCAC	0.04
6	CCCC	0.08
7	CCGC	0.04
8	CCTC	0.04

k	World	$P(W_k)$
9	GCAC	0.06
10	GCCC	0.12
11	GCGC	0.06
12	GCTC	0.06
13	TCAC	0.08
14	TCCC	0.16
15	TCGC	0.08
16	TCTC	0.08

Figure 2: An example of an uncertain sequence and its non-zero possible worlds.

In the presence of data uncertainty, the occurrence of the character of $s[j]$ is now probabilistic. In this paper, we use $P(\hat{s}[j] = \sigma) \in [0,1]$ ($\sigma \in \Lambda$) to denote the probability (confidence) that we observe σ at $\hat{s}[j]$, where $\hat{s}$ is an uncertain sequence defined as follows:

DEFINITION 2.2. ***An uncertain sequence*** *$\hat{s} = \{P(\hat{s}[1] = \sigma), P(\hat{s}[2] = \sigma), ..., P(\hat{s}[n] = \sigma)\}$ is given by a $|\Lambda| \times n$ probability matrix, where $\sigma \in \Lambda$ and $\sum_{\sigma \in \Lambda} P(\hat{s}[j] = \sigma) = 1$, $j \in [1,n]$.*

Based on possible world semantics, an uncertain sequence $\hat{s}$ can be instantiated into a possible world w as a deterministic sequence by choosing the character σ from Λ for each position j of $\hat{s}$ according to $P(\hat{s}[j] = \sigma)$. Given an uncertain sequence $\hat{s}$ with a size of n and a finite alphabet Λ, the number of possible worlds of $\hat{s}$ increases exponentially with n. i.e. $|\Lambda|^n$ in the worst case. Figure 2 shows all 16 non-zero possible worlds derived from $\hat{s}$, where $n = 4$. The probability of a possible world w is denoted as $P(w)$ and $\sum P(w) = 1$. Assuming independence of characters at each position, the probability of w is:

$$P(w) = \prod_{j \in [1,n]} P(\hat{s}[j] = s(w)[j]) \tag{2.1}$$

where $s(w)$ is the instantiated sequence in possible world w.

EXAMPLE 2. *In Figure 2, $P(W_1) = P(s(W_1)[1] = A) \cdot P(s(W_1)[2] = C) \cdot P(s(W_1)[3] = A) \cdot P(s(W_1)[4] = C) = 0.02$.*

Note that we are using a simple model of uncertainty here, where the probabilities of characters at each position of the sequence are independent. If more complex dependencies are used to represent uncertainty, our techniques can be extended. However, we focus on the basic model in this paper. We use the following notation for our core substring matching problem.

DEFINITION 2.3. *The probability that q is a substring of an uncertain sequence $\hat{s}$ is called the* ***substring matching probability*** *and it is denoted as $P(q \sqsubseteq \hat{s})$.*

At the beginning of Section 3 we will describe how this matching probability has an interpretation using the possible worlds model.

DEFINITION 2.4. ***Problem definition:*** *Given a query substring q and an uncertain sequence $\hat{s}$, our main task is to calculate the substring matching probability $P(q \sqsubseteq \hat{s})$.*

Based on the solution of our problem definition, we are able to develop efficient approaches for a number of applications which will be discussed in Section 4.

3 Calculating Substring Matching Probabilities

A simple way to compute the substring matching probability is to enumerate all possible worlds of $\hat{s}$ and sum up the probabilities of possible worlds where $q \sqsubseteq s(w)$, where $s(w)$ is the instantiated sequence in possible world w.

$$P(q \sqsubseteq \hat{s}) = \sum_{w \in W : q \sqsubseteq s(w)} P(w) \tag{3.2}$$

EXAMPLE 3. *In Figure 2, given query substring $q = AC$, $P(q \sqsubseteq \hat{s}) = P(W_1) + P(W_2) + P(W_3) + P(W_4) + P(W_5) + P(W_9) + P(W_{13}) = 0.28$*

Since the number of possible worlds increases exponentially with n, a naive approach is very time costly, as confirmed by our experiments in Section 5.1.

To speed up the calculation, an approximation approach has been proposed in the literature [24]. Equation 3.3 gives an approximate result of the substring matching probability.

$$P(q \sqsubseteq \hat{s}) \approx \sum_{k \in [0,n-m]} \prod_{i \in [1,m]} P(\hat{s}[i+k] = q[i]) \tag{3.3}$$

EXAMPLE 4. *In Figure 2, given query substring $q = AC$, $P(q \sqsubseteq \hat{s}) \approx P(\hat{s}[1] = q[1]) \cdot P(\hat{s}[2] = q[2]) + P(\hat{s}[2] = q[1]) \cdot P(\hat{s}[3] = q[2]) + P(\hat{s}[3] = q[1]) \cdot P(\hat{s}[4] = q[2]) = 0.3 > 0.28$.*

As we can see in the above example, the substring matching probability is overestimated where $P(W_1)$ is counted twice. In addition, the overestimated probability can exceed 1.0. For example, if we change $\hat{s}[1]$ and make it $P(\hat{s}[1] = A) = 1.0$ in the above example. The overestimated probability will be $P(AC \sqsubseteq \hat{s}) = 1.02$. For this reason, the approximate method only gives a score, rather than a true probability. Our experiments in Section 5.2 confirm that this approximation approach yields different results compared to our method, which is able to compute the true probability.

3.1 A Dynamic Programming Approach To avoid the exhaustive enumeration of possible worlds, we propose a dynamic programming approach to efficiently calculate the **exact** substring matching probability. Table 1 summarizes the main notations used in our paper.

DEFINITION 3.1. *For query q and uncertain sequence $\hat{s}$, $q(i) = \{q[1], ..., q[i]\}$ and $\hat{s}(j) = \{\hat{s}[1], ..., \hat{s}[j]\}$, where $i \in [1,m]$ and $j \in [1,n]$.*

$\hat{s}$	Uncertain sequence with size of $\boldsymbol{n}$.
q	Substring to be matched (query substring) over $\hat{s}$, where $\|q\| = \boldsymbol{m}$ and $m \leq n$.
$q[i], \hat{s}[j]$	The ith character of q and the jth character of $\hat{s}$.
$q(i), \hat{s}(j)$	$q(i) = \{q[1], ..., q[i]\}$ and $\hat{s}(j) = \{\hat{s}[1], ..., \hat{s}[j]\}$
$P(\hat{s}[j] = \sigma)$	Probability that we observe character σ at $\hat{s}[j]$, where $\sigma \in \Lambda$ (the alphabet).
$P(q \sqsubseteq \hat{s})$	Probability that q is a substring of $\hat{s}$.
$P_{q(i),j}$	Probability that $q(i)$ is a substring of $\hat{s}(j)$. $P(q \sqsubseteq \hat{s}) = P_{q(m),n}$.

Table 1: Summary of Notation.

DEFINITION 3.2. *$P_{q(i),j}$ is defined as the probability that $q(i)$ is a substring of uncertain sequence $\hat{s}(j)$. Thus, we have $P(q \sqsubseteq \hat{s}) = P_{q(m),n}$*

The main idea of our dynamic programming approach is to split the problem of computing $P_{q(i),j}$ in $\hat{s}(j)$ into subproblems of computing the substring matching probabilities in $\hat{s}(j-1)$. We have four different scenarios that need to be considered: **(1) Forward matching**: we continue to match $q(i-1)$ over $\hat{s}(j-1)$, else **(2) Backward matching**: we move backward and match $q(k)$ over $\hat{s}(j-1)$, where $k \in (i,m)$, else **(3) Tail matching**: we move to the last character of q and match $q(m-1)$ over $\hat{s}(j-1)$, else **(4) Reset**: we reset the current matching position of q and restart matching $q(m)$ over $\hat{s}(j-1)$.

Forward matching: In the step of matching $q(i)$ over $\hat{s}(j)$, if $\hat{s}[j] = q[i]$, then we need to match $q(i-1)$ over $\hat{s}(j-1)$.

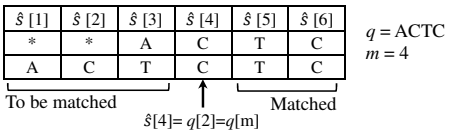
EXAMPLE 5. *Given $q = AGCT$, $\hat{s}$ and $|\hat{s}| = n = 6$. If $\hat{s}[6] = q[4]$, then we need to match AGC over $\hat{s}(5)$ after using the forward matching.*

Backward matching and Tail matching: Alternatively, if $\hat{s}[j] = q[m]$ (i.e. we move backward to match $q[m]$ over $\hat{s}[j]$), then we need to match $q(m-1)$ over $\hat{s}(j-1)$ under the constraint that $j \geq m$. This constraint ensures that there is still enough "room" in $\hat{s}(j)$ for accommodating $q(m)$.

EXAMPLE 6. *Continuing from Example 5, in the step of matching AGC over $\hat{s}(5)$, if $\hat{s}[5] = q[3] = C$, we keep using forward matching. Else, if $\hat{s}[5] = q[4] = T$, then we use tail matching in this step and we need to match AGC over $\hat{s}(4)$ in next step.*

However, if $q[i] = q[m]$, an "ambiguity" will occur. In this case, there are two different interpretations for the step of $\hat{s}(j-1)$: (1) it can be seen as the result of the forward matching of $q(i)$ over $\hat{s}(j)$; (2) it also can be seen as the result of tail matching of $q(m)$ over $\hat{s}(j)$.

EXAMPLE 7. *Given $q = ACTC$, $\hat{s}$ and $|\hat{s}| = n = 6$. Suppose that we have matched C and T in the step of $\hat{s}(6)$ and $\hat{s}(5)$, in the step of matching $q(2) = AC$ over $\hat{s}(4)$, if $\hat{s}[4] = C = q[2] = q[m]$ there are two interpretations for the step of $\hat{s}(3)$: (1) it can be seen as the result of forward matching of AC over $\hat{s}(4)$, thus the current step is to match A over $\hat{s}(3)$. (2) it also can be seen as the result of tail matching ACTC over $\hat{s}(4)$, thus the current step is to match ACT over $\hat{s}(3)$.*



As we can see, if we just consider one of the interpretations, the probability will be underestimated. In the above example, if we only consider to match AC over $\hat{s}(4)$, the case $\hat{s} = ACTCTC$ will not be covered. In fact, this is a key reason why our problem is significantly more complicated than matching in the single-character alphabet case. In our approach, we solve this problem as follows: (1) in the step of matching $q(i)$ over $\hat{s}(j)$, a forward matching is performed; (2) in the next step where matching $q(i-1)$ over $\hat{s}(j-1)$, a backward matching is performed to match $q(m-1)$ over $\hat{s}(j-1)$. The reason for performing a backward matching in $\hat{s}(j-1)$ is to cover the case of the tail matching in $\hat{s}(j)$.

EXAMPLE 8. *Same conditions as in Example 7. In the step of matching $q(2) = AC$ over $\hat{s}(4)$, if $\hat{s}[4] = C$, then (1) in $\hat{s}(4)$, forward matching is used to match AC over $\hat{s}(4)$. (2) In $\hat{s}(3)$, backward matching is used to match ACT over $\hat{s}(3)$.*

However, if $q[m-1] = q[i-1]$, a new "ambiguity" will occur, then backward matching will be postponed to the next step until the entrance condition is satisfied (c.f. Equation 3.6). The general case for backward matching is to match $q(k)$ over $\hat{s}(j-1)$, where $k \in (i,m)$. We call k the ***backward index***. The algorithm that determines k will be discussed shortly.

Reset: The matching will be reset if all the conditions for the above three scenarios are false. Then, we match $q(m)$ over $\hat{s}(j-1)$.

EXAMPLE 9. *Continuing from Example 5, in the step of matching AGC over $\hat{s}(5)$, if $\hat{s}[5] = A$, then we restart matching and match AGCT over $\hat{s}(4)$.*

Overview: Our dynamic programming approach consists of two parts. The first part performs a top-down scan on q (from $q[m]$ to $q[1]$) for computing the backward indices of q. The second part computes the substring matching probability using a bottom-up dynamic programming scheme.

Algorithm 1 Computing-Backward-Index

Input: Substring q with size of m
Output: Backward index array bw

1: $bw[1...m] \leftarrow 0$ // initialize elements in bw to zero
2: **if** $m = 1$ **then**
3: **return**
4: $q[0] \leftarrow \phi$ such that $\forall \sigma \in \Lambda : \sigma \neq \phi$
5: $k \leftarrow 0$ //backward index
6: **for** $i \leftarrow m-1; i \geq 1; i \leftarrow i-1$ **do**
7: **if** $q[i] = q[k]$ **then**
8: $k \leftarrow k-1$
9: **else**
10: $bw[i] \leftarrow k$
11: $k \leftarrow 0$
12: **if** $q[i] = q[m]$ **and** $k = 0$ **then**
13: $k \leftarrow m-1$ // initialize a backward matching

3.1.1 Backward index computation The details of how to compute the backward indices for q, which are stored as an array bw with size of m, are shown in Algorithm 1. Backward matching will be performed in the step of matching $q(i)$ over $\hat{s}(j)$ if $bw[i] \neq 0$. The elements in bw are all initialized to zero in line 1. No backward matching is needed for the trivial case where $m = 1$. For convenience, we define $q[0] = \phi$ such that $\forall \sigma \in \Lambda : \sigma \neq \phi$ (please note that $q[0]$ is not actually stored in q). In line 13, if $q[i] = q[m]$ and $k = 0$, then a backward matching is initialized for the next step $q(i-1)$ (and thus no tail matching for $q(i)$). If $q[i] = q[k]$, we will postpone the backward matching to next step. Otherwise, we set $bw[i] = k$ and reset k (lines 7 – 11).

Figure 3 (a) shows an example of how to compute the backward indices for $q = AGCTCT$ ($m = 6$): (1) k is initialized to 0 for $bw[6]$. (2) $i = 5$: we actually start from $i = m-1 = 5$, where no backward matching is initialized in our example. (3) $i = 4$: because of $q[4] = q[6]$, a backward matching is initialized for $q(i-1) = 3$ with $k = m-1 = 5$. Thus no tail matching is required for $q(4)$. (4) $i = 3$: because of $q[i] = q[k] = C$ ($q[3] = q[5] = C$), we postpone the backward matching to next step. (5) $i = 2$: backward matching will be performed for $q(2)$ with $bw[2] = k = 4$ (additionally, no tail matching for $q(2)$ because $q[bw[2]] = q[6]$, c.f. Equation 3.7). (6) $i = 1$: $bw[1] = k = 0$.

3.1.2 Dynamic programming scheme After the backward indices of input q have been computed, together with the discussion above, we can now calculate the substring matching probability using our dynamic programming scheme. Equation 3.4 is the entry of our approach, where $i = m$ and $j = n$. The two input parameters for the substring matching over $\hat{s}$ are the query substring q and the backward index array bw.

$$P_{q(i),j} = P_{FW}(i,j) + P_{BW}(i,j) + P_{Tail}(i,j) + P_{Reset}(i,j) \tag{3.4}$$

where

$$P_{FW}(i,j) = P_{q(i-1),j-1} \times P(\hat{s}[j] = q[i]) \tag{3.5}$$

Equation 3.6 Entrance condition: $bw[i] \neq 0$ and $j \geq bw[i]$

$$P_{BW}(i,j) = P_{q(bw[i]-1),j-1} \times P(\hat{s}[j] = q[bw[i]]) \tag{3.6}$$

Equation 3.7 Entrance condition: $q[i] \neq q[m]$ and $q[bw[i]] \neq q[m]$ and $j \geq m$

$$P_{Tail}(i,j) = P_{q(m-1),j-1} \times P(\hat{s}[j] = q[m]) \tag{3.7}$$

$$P_{Reset}(i,j) = P_{q(m),j-1} \times (1 - P(\hat{s}[j] = q[i]) - p_{bw} - p_{tail}) \tag{3.8}$$

where, $p_{bw} = P(\hat{s}[j] = q[bw[i]])$ if entering Equation 3.6; $p_{bw} = 0$ otherwise. $p_{tail} = P(\hat{s}[j] = q[m])$ if entering Equation 3.7; $p_{tail} = 0$ otherwise.

The recursion termination conditions are:

$$P_{q(0),j} = 1 \text{ and } P_{q(i),j} = 0, \forall i > j \tag{3.9}$$

The above equations are further explained as follows. After entering Equation 3.4 with $i = m$ and $j = n$, forward matching is computed via Equation 3.5 if $\hat{s}[j] = q[i]$. The backward index $bw[i]$ is retrieved in Equation 3.6, if the entrance condition is met. The condition $j \geq bw[i]$ ensures there is still enough "room" in $\hat{s}(j)$ for accommodating $q(bw[i])$. No tail matching for $q(i)$ if $q[i] = q[m]$ or $q[bw[i]] = q[m]$ (Equation 3.7) where an "ambiguity" will occur. The matching will be reset via Equation 3.8 if the other three scenarios are false. These equations are calculated recursively. The recursion termination conditions are shown in Equation 3.9.

Figure 3 (b) shows an example of how to use our dynamic programming approach to match $q = AGCTCT$ over $\hat{s}(n)$, where $m = 6$ and $n = 10$. For clarity, $q(i)$ is explicitly shown at each iteration. The characters of q that have been "consumed" are underlined. For example, $P_{AGCT\underline{CT},5}$ represents $P_{q(4),5}$. Theoretically, the equations can be used as a top-down approach. However, this approach leads to repeated calculations of internal results. For example, in Figure 3, $P_{AGCT\underline{CT},5}$ is used by both $P_{AGCT\underline{CT},6}$ and $P_{AG\underline{CTCT},6}$. Instead, as a dynamic programming method, we use a bottom-up approach. As shown in Figure 3 (b), we compute the matching probability **column by column** from $j = 1$ to $j = n$. For each column j, the rows we need to compute in a bottom-up manner are between $[i_{min}(j), i_{max}(j)]$, where $i_{min}(j) = max(m-(n-j),1)$ and $i_{max}(j) = min(j,m)$. For example, for column $j = 5$, $i_{min}(j = 5) = 1$ and $i_{max}(j = 5) = 5$. At

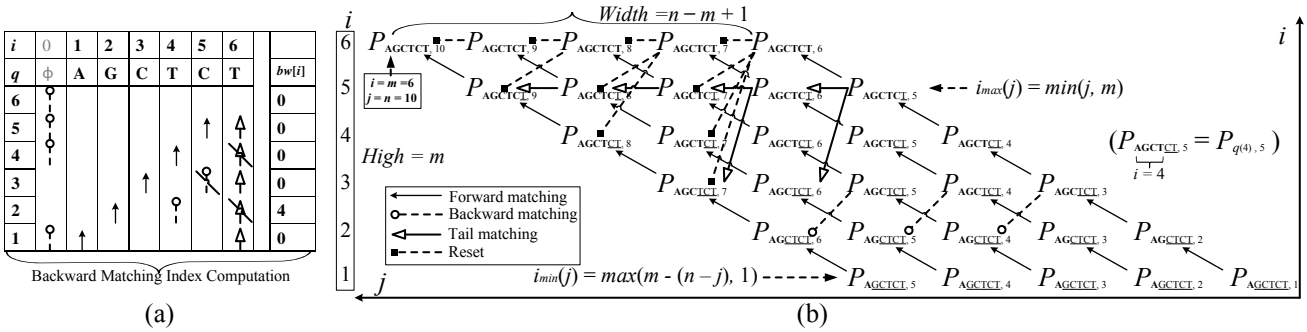


Figure 3: Details of how to calculate $P_{q(6),10}$ ($q = AGCTCT$) using our dynamic programming approach: (a) A top-down scan on q to compute the backward index array bw. (b) A bottom-up dynamic programming scheme to compute $P_{q(6),10}$.

each iteration, we store internal results for further calculations until we reach the upper-leftmost node.

The width for each row is $n-m+1$ with m rows in total. Thus the number of computed nodes is $-m^2+n\cdot m+m$. It is a quadratic function and reaches its peak for $m = n/2$. We now state a key theorem for the computational complexity of our method to compute the substring matching probability.

THEOREM 1. *Both the time complexity and space complexity for calculating $P(q \sqsubseteq \hat{s}) = P_{q(m),n}$ are $O(n)$.*

Proof. To compute the backward index array of q, we need to perform one scan on q and store the indices in bw ($|bw| = m$). For the dynamic programming scheme, the nodes we need to compute and store are parallelogram in shape with $width = n-m+1$ and $height = m$. Thus, it requires $O(m+(n-m+1)\cdot m) = O(m\cdot n - m^2 + 2\cdot m) = O(m\cdot n)$ time and space. Since m is assumed to be small and thus can be bounded by a constant c, the overall complexity is $O(n)$ (a similar analysis was given in [6]).

The Example that shows how to use our approach to calculate $P(q \sqsubseteq \hat{s}) = P_{q(6),10}$ over $\hat{s}$ from Figure 1, where $q = AGCTCT$, can be found on [15].

Comment: Note that the internal results $P_{q(i),j}$ (where $i < m$) computed by our algorithm are used for computing $P_{q(m),j}$ only. That is, given $q = \{q[1],...,q[m]\}$ and $q' = \{q[1],...,q[m']\}$, where $m > m'$, $P_{q(m'),j} \neq P_{q'(m'),j}$. This is due to the consecutiveness (no gap) constraint of substrings.

4 Applications and Extensions

In this section, we will sketch a number of applications and extensions that leverage our solution for computing the substring matching probability.

4.1 Computing the Frequentness Probabilities of Substrings

Given a collection of uncertain sequences $S_{DB} = \{\hat{s}_1, \hat{s}_2, ..., \hat{s}_{|S_{DB}|}\}$, we may call S_{DB} an uncertain sequence database. In the context of deterministic databases, given a minimum support threshold $minSupp$, a string q is called a frequent substring if $support(q) \geq minSupp$. Similarly, in uncertain sequence databases, a substring is called probabilistic frequent if $P_{\geq minSupp}(q) \geq \tau$, where $P_{\geq minSupp}(q)$ denotes the probability that q is a substring of at least $minSupp$ uncertain sequences in S_{DB} (called the **frequentness probability** of q) , and τ is the probabilistic threshold. In order to check whether q is probabilistic frequent, we need to compute $P_{\geq minSupp}(q)$.

The important step of this task is to compute $\{P(q \sqsubseteq \hat{s}_1), P(q \sqsubseteq \hat{s}_2), ..., P(q \sqsubseteq \hat{s}_{|S_{DB}|})\}$, which can be obtained by our techniques proposed in Section 3.1. After that, this task can be seen as an analogue of uncertain frequent itemset mining [6] or uncertain timestamp sequence mining [16] (where we set $gap = inf$). The formula is as follows:

(4.10)
$$P_{\geq i,j}(q) = P_{\geq i-1,j-1}(q)\cdot P(q \sqsubseteq \hat{s}_j) + P_{\geq i,j-1}\cdot(1 - P(q \sqsubseteq \hat{s}_j))$$

where $P_{\geq i,j}(q)$ is the probability that q have a support of at least i in the first j uncertain sequences. The entry case is $i = minSupp$ and $j = |S_{DB}|$. As we can see, computing $P(q \sqsubseteq \hat{s}_j)$ is an essential part of this task.

4.2 Mining Probabilistic Frequent Substring Patterns

Once we know how to decide whether a substring is probabilistic frequent $P_{\geq minSupp}(q) \geq \tau$, a further step can be employed that uses an enumeration framework for mining all the probabilistic frequent substrings from S_{DB}. We leave the merits of different frameworks as an open question. Fortunately, as shown in [6], the support anti-monotonicity (Apriori rule) still holds for uncertain data. That is, if q is not probabilistic frequent then any proper superset of q can not be probabilistic frequent. Thus, standard pattern enumeration frameworks can be used to output all probabilistic frequent substring patterns, such as Apriori enumeration, or set enumeration trees.

4.3 Computing the Similarity between Two Uncertain Sequences Computing the similarity between two DNA sequences or protein sequences is a common task in biological research and can be used for similarity and clustering. In some contexts, the two sequences may be uncertain - such as the case when the sequences correspond to motifs represented by a position-frequency matrix, describing binding probabilities of a protein. A popular approach for similarity is to use k-mer spectrum comparison [24]. A k-mer is a short substring of length k and a biological sequence can be translated into a vector, whose features correspond to k-mers and whose feature values correspond to the probability of the k-mers occurring in the sequence.

Given an alphabet Λ and k and uncertain sequences $\hat{s}_1$ and $\hat{s}_2$, we calculate $P(q \sqsubseteq \hat{s}_1)$ and $P(q \sqsubseteq \hat{s}_2)$, where $q \in \binom{|\Lambda|}{k}$. The results will be two vectors describing the probability distributions of k-mers within each sequence. Standard vector similarity measures, such as cosine similarity, can then be used to calculate the similarity of these vectors.

4.4 Matching Subsequences without Consecutiveness Constraint According to Definition 2.1, the characters of q in s are required to be all consecutive (no gaps). However, one might wish to relax this constraint for some applications.

DEFINITION 4.1. *q is called a **subsequence** of s iff q can be derived from s by deleting finite characters without changing the order of the remaining characters.*

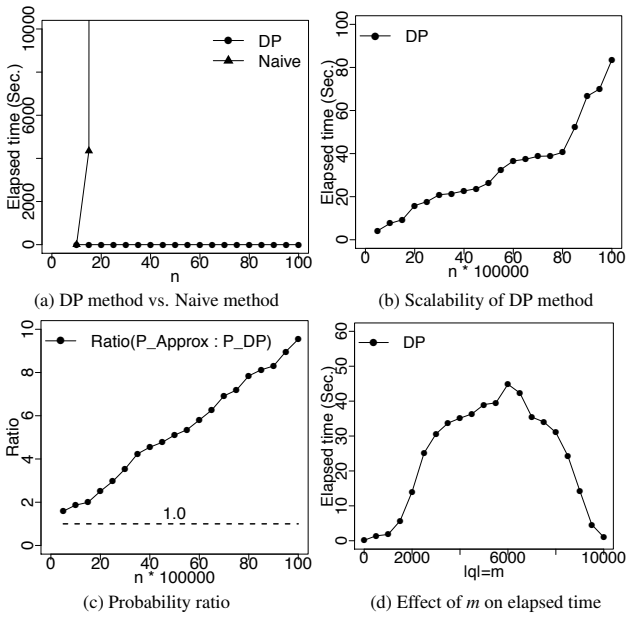
For example, $q = GC$ is a subsequence of $s = GACT$ but not a substring of s. In the context of uncertain data, the subsequence matching is a task to compute the probability that q is a subsequence of $\hat{s}$ which is denoted as $P(q \subseteq \hat{s})$.

Without the consecutiveness constraint, the computation of subsequence matching probability becomes more straightforward. Using similar notations to Section 3.1, we can calculate $P(q \subseteq \hat{s}) = P_{q(m),n}$ as follows. In the step of matching $q(i)$ over $\hat{s}(j)$, if $\hat{s}[j] = q[i]$, we need to match $q(i-1)$ over $\hat{s}(j-1)$. If $\hat{s}[j] \neq q[i]$, because we do not have a consecutiveness constraint in a subsequence, we now can rematch partial of q (i.e. $q(i)$ where $i < m$) anytime. That is, we can match $q(i)$ over $\hat{s}(j-1)$. This technique has also been used in previous work on probabilistic top-k queries [25]. The subsequence matching probability is computed by:

(4.11)
$$P_{q(i),j} = P_{q(i-1),j-1} \cdot P(\hat{s}[j] = q[i]) + P_{q(i),j-1} \cdot P(\hat{s}[j] \neq q[i])$$

Where $P(q \subseteq \hat{s}) = P_{q(m),n}$ is the entry case.

EXAMPLE 10. *Given $q = AGCT$, $\hat{s}$ and $|\hat{s}| = n = 6$. After we match $q[4] = T$ and $q[3] = C$ in $\hat{s}(6)$ and $\hat{s}(5)$. If $q[2] \neq \hat{s}[4]$, we can rematch $q(2) = AG$ over $\hat{s}(3)$, which is not allowed in the substring matching due to the consecutiveness constraint.*



(a) DP method vs. Naive method (b) Scalability of DP method

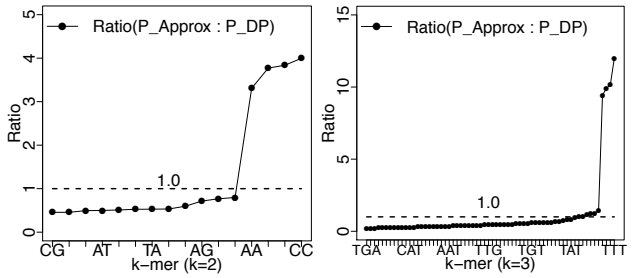
(c) Probability ratio (d) Effect of m on elapsed time

Figure 4: Synthetic datasets: (a) Dynamic programming method vs. Naive method. (b) Scalability of DP method. (c) Probability ratio: "P_DP" denotes the exact probability calculated by our dynamic programming approach; "P_Approx" denotes the approximate probability calculated by Equation 3.3. (d) Effect of m on elapsed time.

5 Experiments

We first consider large synthetic datasets to test the efficiency of our dynamic programming approach for the substring matching probability. Then, we use real-world uncertain DNA data to further explore the behaviour of our algorithm. All experiments were performed on an Intel Core i7 3.4GHz machine with 32GB main memory.

5.1 Evaluation on Synthetic Data We use synthetic uncertain sequences to test the efficiency of our dynamic programming approach. Our experiments measure the (average) time needed to compute the substring matching probability, across different choices of q and $\hat{s}$. The size of the uncertain sequence $\hat{s}$ varies from $n = 10$ to $n = 10^7$ and $\Lambda = \{A, C, G, T\}$. The probability $P(\hat{s}[j] = \sigma)$ is randomly drawn from $[0, 1]$, where $j \in [1, n]$, $\sigma \in \Lambda$ and $\sum_{\sigma \in \Lambda} P(\hat{s}[j] = \sigma) = 1$. All results are the average of ten runs. The default size of the query substring q is $m = 10$, unless stated otherwise. The characters in q are randomly chosen from Λ. Abbreviations used in our figures: (1) Naive: naive approach of matching probability calculation (c.f. Equation 3.2); (2) Approx: the approximation approach of [24] (c.f. Equation 3.3); (3) DP: our dynamic programming approach.



(a) Probability ratios for 2-mers (b) Probability ratios for 3-mers

Figure 5: Evaluation on a real-world uncertain DNA database. The probability ratios of queries are sorted in the ascending order.

5.1.1 Scalability Figure 4 (a) compares the elapsed time of our dynamic programming approach against the naive method. The elapsed time of the naive method increases exponentially when the size of the uncertain sequence increases. The elapsed time of the naive method is out of the figure's maximum scale when $|\hat{s}| = n = 20$. In fact, the total number of possible worlds has already reached 4^{20} by that point. Thus the naive method is not practical. The scalability of our proposed dynamic approach is illustrated in Figure 4 (b). The performance of our proposed approach is highly promising: as n increases, the running time grows in a linear trend as in line with our time complexity theorem (c.f. Theorem 1).

Figure 4 (c) shows the ratio between the exact probability ("P_DP") calculated by our dynamic programming approach and the approximate probability ("P_Approx"), which is a likelihood score, calculated by Equation 3.3, using the same group of synthetic data above. As we can see, the difference between the true probability and the approximate probability becomes more significant as n increases. The approximate probabilities are considerably overestimated in our synthetic datasets.

5.1.2 Effect of m on Elapsed Time The elapsed time for calculating $P_{q(m),10^4}(O)$ is shown in Figure 4 (d), where m varies from 1 to 10^4 with an increment of 500. As we can see, the elapsed time reaches a peak at around $6000 \approx n/2 = 5000$. The curve in the figure is not completely symmetrical. For example, the elapsed time at $m = 10000$ is slightly higher than $m = 1$. The reason is that in the first part of our dynamic programming approach, we need to scan q and compute the backward index array. As m increases, the right half of the figure takes slightly more time for computing the backward index array than the left half.

5.2 Evaluation on Real Data A real-world uncertain DNA sequence database [7] was next used to explore the differences between our method and the approximation approach for substring matching from [24]. The database contains 476 uncertain sequences where $\Lambda = \{A, C, G, T\}$. Each uncertain sequence corresponds to a transcription factor (protein/motif) which can bind to a DNA sequence. The uncertainty arises from the fact that each protein is capable of binding to different sites on the DNA and a consensus description of a protein's binding preferences requires uncertainty at each position. The total length of sequences is 4720 and each uncertain sequence is relatively short (on average size 10). The elapsed time of experiments on this dataset is negligible for our approach. However, our primary aim here is to compare the output of our approach with the output of the approximation method of equation 3.3, as follows. We enumerated all combinations of k-mers ($k = 2$ and $k = 3$) and got 16 2-mers and 64 3-mers. Using each k-mer (as a query substring q), we computed the substring matching probabilities for all 476 uncertain sequences and then took the average. The top matched k-mers (substrings) are shown in Figure 6.

Rank	2-mer	Probability	ApproxProb	3-mer	Probability	ApproxProb
1	AA	0.552	1.825	TAA	0.289	0.091
2	AT	0.541	0.270	AAT	0.277	0.091
3	TA	0.510	0.270	ATT	0.233	0.090
4	TG	0.440	0.201	TTA	0.230	0.090
5	GA	0.430	0.256	GGA	0.156	0.085

(a)

Rank	2-mer	ApproxProb	Probability	3-mer	ApproxProb	Probability
1	AA	1.825	0.552	AAA	1.408	0.149
2	TT	1.648	0.429	TTT	1.216	0.102
3	GG	1.441	0.382	GGG	1.103	0.108
4	CC	1.311	0.328	CCC	0.983	0.099
5	AT	0.270	0.541	TAA	0.091	0.289

(b)

Figure 6: The 2-mers and 3-mers that have highest substring matching probabilities discovered from the uncertain DNA sequence database. (a) Ranked by our dynamic programming approach. (b) Ranked by the approximation approach.

Figure 6 shows that the two methods do not rank the k-mers in the same way. For example, *TAA* is the top ranked 3-mer by our method, yet it is ranked number 5 by the approximation approach. The approximation approach ranks *AAA* as the top 3-mer, yet it is not ranked in the top five by our approach. As we can see, the two approaches return significantly different rankings for both 2-mers and 3-mers. It can be observed that the approximation approach (c.f. Equation 3.3) favors k-mers consisting of single character. Using Spearman's rank correlation coefficient ρ to compare the two rankings we found i) for 2-mers, $\rho = 0.36$ and ii) for 3-mers, $\rho = 0.41$. Both cases indicating only a mild positive correlation between the rankings output by the two methods.

Figure 5 compares the exact probabilities ("P_DP") calculated by our dynamic programming approach against approximate probabilities ("P_Approx") calculated by Equation 3.3 for 2-mers and 3-mers in this database. The proba-

bility ratios of queries are sorted in the ascending order. The probability for each query is the average of 476 uncertain sequences. In Figure 5 (a), the matching probabilities of 2-mers: *AA*, *GG*, *TT* and *CC* are overestimated by the approximation approach, while the matching probabilities of other 2-mers are underestimated. In Figure 5 (b), the matching probabilities of *TCT*, *AAA*, *CCC*, *GGG* and *TTT* are among those most overestimated. As we can see, the approximation approach differs in its output for this type of real-world data.

Based on Figures 5 and 6 we can make the following conclusions: i) our exact approach and the approximation approach provide different rankings for the substring matching probability of *k*-mers, ii) our method provides a true and theoretically justified probability calculation, whereas the approximation does not, iii) using the approximation approach could lead to different (and possibly erroneous) conclusions. In summary, our exact approach has great potential to be used as a replacement for the approximate approach in this type of application.

6 Conclusions

In this paper, we have formulated and studied the problem of substring matching in uncertain sequences. We proposed a dynamic programming approach for computing the substring matching probability with linear time complexity. We further investigated a number of applications and extensions based on our solution of substring matching. We provided experimental evidence that demonstrated the scalability of our approach and also contrasted its exact output, against that of an approximate algorithm. Our results contribute towards a foundation for adapting classic sequence mining methods to deal with uncertain data.

References

[1] C. Aggarwal, Y. Li, J. Wang, and J. Wang, *Frequent pattern mining with uncertain data*, in SIGKDD, ACM, 2009.

[2] C. C. Aggarwal and P. S. Yu, *A survey of uncertain data algorithms and applications*, TKDE, (2009).

[3] R. Agrawal, C. Faloutsos, and A. Swami, *Efficient similarity search in sequence databases*, Springer, 1993.

[4] R. Agrawal and R. Srikant, *Mining sequential patterns*, in ICDE, IEEE, 1995, pp. 3–14.

[5] J. Ayres, J. Flannick, J. Gehrke, T. Yiu, et al., *Sequential pattern mining using a bitmap representation*, in KDD, 2002, pp. 429–435.

[6] T. Bernecker, H. Kriegel, M. Renz, F. Verhein, and A. Zuefle, *Probabilistic frequent itemset mining in uncertain databases*, in SIGKDD, ACM, 2009, pp. 119–128.

[7] J. Bryne, E. Valen, M. Tang, T. Marstrand, O. Winther, I. Da Piedade, A. Krogh, B. Lenhard, and A. Sandelin, *Jaspar, the open access database of transcription factor-binding profiles: new content and tools in the 2008 update*, Nucleic acids research, (2008).

[8] T. Calders, C. Garboni, and B. Goethals, *Efficient pattern mining of uncertain data with sampling*, in PAKDD, Springer, 2010, pp. 480–487.

[9] C. Chui and B. Kao, *A decremental approach for mining frequent itemsets from uncertain data*, in PAKDD, 2008.

[10] C. Chui, B. Kao, and E. Hung, *Mining frequent itemsets from uncertain data*, in PAKDD, Springer, 2007.

[11] C. Faloutsos, M. Ranganathan, and Y. Manolopoulos, *Fast subsequence matching in time-series databases*, ACM, 1994.

[12] T. Ge and Z. Li, *Approximate substring matching over uncertain strings*, in PVLDB, 2011.

[13] M. Hua, J. Pei, W. Zhang, and X. Lin, *Efficiently answering probabilistic threshold top-k queries on uncertain data*, in ICDE, IEEE, 2008, pp. 1403–1405.

[14] ———, *Ranking queries on uncertain data: A probabilistic threshold approach*, in SIGMOD, ACM, 2008, pp. 673–686.

[15] Y. Li, J. Bailey, L. Kulik, and J. Pei, *Supplementary document*. `http://www.csse.unimelb.edu.au/~jbailey/sdm14_suppdoc.pdf`.

[16] ———, *Mining probabilistic frequent spatio-temporal sequential patterns with gap constraints from uncertain databases*, in ICDM, IEEE, 2013.

[17] Y.-S. Moon, K.-Y. Whang, and W.-S. Han, *General match: a subsequence matching method in time-series databases based on generalized windows*, in SIGMOD, ACM, 2002.

[18] M. Muzammal and R. Raman, *Mining sequential patterns from probabilistic databases*, in PAKDD, 2011.

[19] J. Pei, J. Han, B. Mortazavi-Asl, J. Wang, H. Pinto, Q. Chen, U. Dayal, and M. Hsu, *Mining sequential patterns by pattern-growth: The prefixspan approach*, in ICDE, 2004.

[20] R. Srikant and R. Agrawal, *Mining sequential patterns: Generalizations and performance improvements*, in EDBT, Springer, 1996, pp. 1–17.

[21] L. Sun, R. Cheng, D. Cheung, and J. Cheng, *Mining uncertain data with probabilistic guarantees*, in SIGKDD, ACM, 2010, pp. 273–282.

[22] Y. Tong, L. Chen, Y. Cheng, and P. Yu, *Mining frequent itemsets over uncertain databases*, PVLDB, (2012).

[23] L. Wan, C. Ling, and C. Zhang, *Mining frequent serial episodes over uncertain sequence data*, in EDBT, 2013.

[24] M. Xu and Z. Su, *A novel alignment-free method for comparing transcription factor binding site motifs*, PloS one, (2010).

[25] K. Yi, F. Li, G. Kollios, and D. Srivastava, *Efficient processing of top-k queries in uncertain databases with x-relations*, in TKDE, IEEE, 2008, pp. 1669–1682.

[26] M. Zaki, *Spade: An efficient algorithm for mining frequent sequences*, Machine Learning, (2001), pp. 31–60.

[27] Q. Zhang, F. Li, and K. Yi, *Finding frequent items in probabilistic data*, in SIGMOD, ACM, 2008, pp. 819–832.

[28] Z. Zhao, D. Yan, and W. Ng, *Mining probabilistically frequent sequential patterns in uncertain databases*, in EDBT, ACM, 2012, pp. 74–85.