

Distributed Sequential Pattern Mining in Large Scale Uncertain Databases

Jiaqi Ge^(✉) and Yuni Xia

Department of Computer and Information Science,
Indiana University Purdue University, Indianapolis, IN 46202, USA
{jiaqge,yxia}@cs.iupui.edu

Abstract. While sequential pattern mining (SPM) is an import application in uncertain databases, it is challenging in efficiency and scalability. In this paper, we develop a dynamic programming (DP) approach to mine probabilistic frequent sequential patterns in distributed computing platform Spark. Directly applying the DP method to Spark is impractical because its memory-consuming characteristic may cause heavy JVM garbage collection overhead in Spark. Therefore, we design a memory-efficient distributed DP approach and use an extended prefix-tree to save intermediate results efficiently. The extensive experimental results in various scales prove that our method is orders of magnitude faster than straight-forward approaches.

Keywords: Uncertain databases · Sequential pattern mining · Distributed computing

1 Introduction

Sequential pattern mining (SPM) is one of the most important applications in data mining. It is widely used to analyze customer behaviors in market-basket databases. For example, online shopping websites usually collect customer purchasing records in databases where sequential patterns are mined to reveal buying habits of consumers. However, in many real applications, events occurring in a sequence may be uncertain for many reasons. For instance, data collected by sensors are inherently noisy; in privacy protection applications, artificial noises are added deliberately; data modeling techniques such as classifications may also produce indeterministic results [1].

Example 1. Consider an online travel website. To increase sales, a large group of customers are analyzed in order to discover sequential patterns of user-interested products. These patterns are useful in intelligenet marketing. For example, by providing a special hotel offer to a customer who booked a late-night flight, the website is able to encourage hotel purchases.

Figure 1(a) records user preferences to various travel products. For example, in the session B , the customer is first attracted by a rental car and then shows

Session	timestamp	Products	Prob.
A	1	flight	0.7
A	2	flight, hotel	0.6
A	3	hotel	0.8
B	1	car	1.0
B	2	hotel	0.7

(a) uncertain sequence database

world	Sequence	Prob.
1	<(flight);> <(flight, hotel);> <(hotel)>	0.336
2	<(flight);><(flight, hotel)>	0.084
3	<(flight);><(hotel)>	0.224
4	<(flight)>	0.056
5	<(flight, hotel);><(hotel)>	0.144
6	<(flight, hotel)>	0.036
7	<(hotel)>	0.096
8	\emptyset	0.024

(b) Possible worlds of S_A

Fig. 1. Example application of an uncertain sequence database

interests in a hotel with a probability of 0.7. Here user interests are estimated by posterior probabilities from Naïve Bayesian models which takes features, such as how long a customer stays on a page and so on, into consideration. The website uses a database that represents each visiting session as a single sequence, also shown in Fig. 1(a).

1.1 Problem Statement

The uncertain model applied in this paper is based on possible world semantics with existential *uncertain events*.

Definition 1. An uncertain event is an event e whose presence in a sequence d is defined by an existential probability $P(e \in d) \in (0, 1]$.

Definition 2. An uncertain sequence d is an ordered list of uncertain events. An uncertain sequence database is a collection of uncertain sequences.

Definition 3. A sequential pattern $s = \langle s_1, \dots, s_n \rangle$ is an ordered list of itemsets where an itemset $s_i \in s$ is also called an element of s .

In certain databases, a sequential pattern $s = \langle s_1, \dots, s_n \rangle$ is *supported* by a sequence $d = \langle e_1, \dots, e_m \rangle$, denoted by $s \preceq d$, if there exists n integers $1 \leq k_1 < \dots, k_n \leq m$ that have $s_i \subseteq e_{k_i}$ for $i \in [1, n]$. A sequential pattern is *frequent* if at least τ_s sequences support it, where τ_s is a user-specified threshold. However, in an uncertain database D , the support of a pattern s is uncertain. We define *probabilistic frequent sequential patterns* (p-FSP) as follows:

Definition 4 (Probabilistic Frequent Sequential Pattern). A sequential pattern s is a probabilistic frequent sequential pattern (p-FSP) if its probability of being frequent is at least τ_p , denoted by $P(\text{sup}(s) \geq \tau_s) \geq \tau_p$.

Here $\text{sup}(s)$ is the support of s in D and τ_p is the user-defined minimum confidence in the frequentness of a sequential pattern. We are now able to specify the uncertain SPM problem as: *Given τ_s, τ_p , find all p-FSPs in D .*

In an uncertain database, sequences are often assumed to be mutually independent, which is also known as the *tuple-level independence* [2, 7]. Therefore, the overall support $\text{sup}(s)$ in D is a sum of uncertain supports in every single

sequence. And the probabilistic support of s in an uncertain sequence d_i can be modeled by a Bernoulli random variable $X_i \sim B(1, p_i)$, where $p_i = P(s \preceq d_i)$ is the probability that d_i supports s .

When the size of D grows, we can approximate the distribution of $\text{sup}(s)$ by the Gaussian distribution in Eq. (1), according to *the central limit theory*.

$$\text{sup}(s) \xrightarrow{|D| \rightarrow \infty} \mathcal{N}\left(\sum_{i=1}^{|D|} p_i, \sum_{i=1}^{|D|} p_i * (1 - p_i)\right) \quad (1)$$

We are now able to compute the *approximate frequentness probability* of s by:

$$P(\text{sup}(s) \geq \tau_s) = 1 - P(\text{sup}(s) \leq \tau_s - 1) = 1 - \Phi\left(\frac{\tau_s - 1 - \mu}{\sigma}\right) \quad (2)$$

And s is a p-FSP if $P(\text{sup}(s) \geq \tau_s) \geq \tau_p$.

Now the key issue of uncertain SPM is the computation of support probabilities. We use possible world semantics to interpret uncertain sequences. Here a possible world is a *certain sequence* instantiated by generating every event according to its existential probability. Each uncertain probability $P(e \in d)$ derives two possible worlds per sequence: One possible world in which event e exists in sequence d , and the other possible world where e does not exist. Therefore, the number of possible worlds increases exponentially in the number of events.

Each possible world w is associated with an existential probability $P(w)$. Figure 1(b) shows all possible worlds derived from sequence S_A in Fig. 1(a). For example, in world 2, the customer in session A is first attracted by a *flight* product and then interested in a *flight-hotel* package. Afterwards, the customer has no interests towards another *hotel* any more.

Uncertain events in a sequence are also assumed to be independent of each other [4, 13], since they are often observed independently in real world applications. Therefore, we can compute the existential probability of a possible world w by Eq. (3).

$$P_e(w) = \prod_{e \in d} P(e \in d) * \prod_{e \notin d} (1 - P(e \in d)) \quad (3)$$

For example, the existential probability of world 2 in Fig. 1(b) is $P(w_2) = 0.7 * 0.6 * (1 - 0.8) = 0.084$.

In this paper, we focus on the problem of calculating support probabilities in large scale databases and extracting all p-FSPs. Meanwhile, in order to mine highly scalable databases, we extend the Apriori-like framework of uncertain SPM to Spark [12] which is a distributed computing platform allowing us to load data into a cluster's memory and query it repeatedly.

1.2 Contribution

In this paper, we propose a **D**istributed **S**equential **P**attern (DSP) mining algorithm in large scale uncertain databases. Our main contributions are summarized

as follows: (1) We propose a SPM framework for mining large scale uncertain databases in Spark. (2) We present a dynamic programming method to compute support probabilities in linear time. (3) We propose a memory-efficient distributed dynamic programming approach in Spark and design a new data structure to save intermediate results efficiently. (4) Extensive experiments conducted in various scales shows that our algorithm is orders of magnitude faster than both direct extension and existing works.

2 Related Works

Uncertain data mining has been an active area of research recently. Many traditional database and data mining techniques have been extended to be applied to uncertain databases [2]. Specifically, Muzammal and Raman first define the SPM problem in probabilistic database [10]; and Zhao et al. define probabilistic frequent sequential patterns in possible world model and propose their uncertain SPM algorithms [13, 14]. Li et al. introduce a dynamic programming approach to mine sequential patterns in a specific spatial-temporal uncertain model [8]. Wan et al. [11] propose a dynamic programming algorithm of mining frequent serial episodes within one uncertain sequence. However, all the above mentioned methods can only be executed in a single machine and may have scalability issues in mining large databases.

Chen et al. extend the classic SPAM algorithm to its MapReduce version SPAMC [5]. Miliaraki et al. propose a gap-constraint frequent sequence mining algorithm in MapReduce [9]. These algorithms are applied in the context of deterministic data, while our work aims to solve large scale uncertain SPM problems. An iterative MapReduce implementation of uncertain SPM in [6] is somehow close to our work; however, it has a quadratic complexity of support probability computation, and the time cost of that in our algorithm is linear.

3 Uncertain SPM Framework in Spark

First of all, we define the following two types of sequential pattern extension.

Definition 5 (Item-extended Pattern). *An item-extended pattern s is a sequential pattern generated by adding a new item i to the first element of another sequential pattern s' , denoted by $s = \{i\} \cup s'$.*

Definition 6 (Sequence-extended Pattern). *A sequence-extended pattern s is a sequential pattern generated by adding a new itemset $\{i\}$ to another sequential pattern s' as its first element, denoted by $s = \{i\} + s'$.*

For example, let $s' = \langle (b)(d) \rangle$, then $s_1 = \langle (a, b)(d) \rangle$ is an item-extended pattern of s' and $s_2 = \langle (a)(b)(d) \rangle$ is sequence-extended from s' . The Apriori property of p-FSPs in Lemma 1 allows us to prune a pattern if it is extended from a pattern which is not a p-FSP. [14]

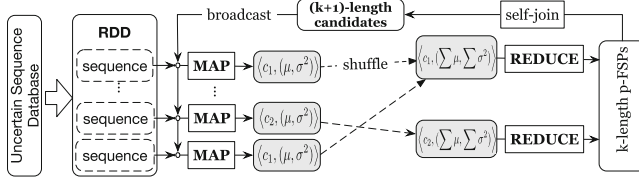


Fig. 2. A framework of DSP in Spark

Lemma 1. *If s is extended from s' and s is a p -FSP, then s' is also a p -FSP.*

Figure 2 shows the Apriori-like uncertain SPM framework of our DSP algorithm in Spark.

An uncertain sequence database $D = \{d_1, \dots, d_n\}$ is abstracted by an RDD [12] in Spark. These sequences are allocated to a cluster of machines and can be processed in parallel.

Map. A map function is used to compute support probabilities. A set of candidate patterns are broadcasted to all the mappers. For each candidate pattern c , the map function first computes the support probability $p_i = P(c \preceq d_i)$ in the uncertain sequence d_i , then it emits a key-value pair as $\langle c, (\mu_i, \sigma_i^2) \rangle$, if $p_i > 0$. The key field here is the pattern c ; the composite value field contains both mean μ_i and variance σ_i^2 of the Bernoulli distributed probabilistic support $X_i \sim \mathcal{B}(1, p_i)$. The key-value pairs are designed to be associative and commutative, so that Spark can aggregate them first in local machines to minimize network usage in shuffling.

Reduce. Pairs with the same key are shuffled to one reducer. In a reduce function, it computes the approximate frequentness probability for each candidate by Eq. (2). All candidates with $P(\text{sup}(c) \geq \tau_s) \geq \tau_p$ are saved to a set of k -length p-FSPs, denoted by S_k .

Self-join. we self-join all k -length p-FSPs in S_k to generate a set of $(k+1)$ -length candidate patterns in C_{k+1} . Let s_1 and s_2 be two p-FSPs in S_k . Suppose s'_1 is the pattern generated by removing the first item i in s_1 and s'_2 is the pattern generated by removing the last item of s_2 . If $s'_1 = s'_2$, we join s_1 and s_2 , denoted by $s_1 \bowtie s_2$, to generate a $(k+1)$ -length candidate c according to the following rules: If s_1 is sequence-extended, $c = \{i\} + s_2$; if s_1 is item-extended, $c = \{i\} \cup s_2$. For example, let $s_1 = \langle (a)(bc) \rangle$, $s_2 = \langle (bc)(d) \rangle$ and $s_3 = \langle (c)(de) \rangle$, then $s_1 \bowtie s_2 = \langle (a)(bc)(d) \rangle$; while $s_2 \bowtie s_3 = \langle (bc)(de) \rangle$.

Stop Criterion. If either S_k or C_{k+1} is empty, we terminate the mining process; otherwise, C_{k+1} is broadcasted to all map functions for the next iteration.

4 A Distributed Dynamic Programming Approach

4.1 Dynamic Programming in Support Probability Computation

In this section, we propose a DP method to compute support probabilities. The key to our approach is to consider it in terms of sub-problems. Here we first define $P_{i,j}$ in Definition 7.

Definition 7. $P_{i,j} = P(s_i^n \preceq d_j^m)$ is the probability that s_i^n is supported by d_j^m , where $s_i^n = \langle s_i, \dots, s_n \rangle$ is a subsequence of a sequential pattern s and $d_j^m = \langle e_j, \dots, e_m \rangle$ is a subsequence of an uncertain sequence d .

Therefore, $P(s \preceq d) = P_{1,1}$. The idea in our approach is to split the problem of computing $P_{i,j}$ into sub-problems $P_{i,j+1}$ and $P_{i+1,j+1}$. And this can be achieved as follows: in condition of $s_i \subseteq e_j$, $P_{i,j}$ is equal to the probability that s_{i+1}^n is supported by d_{j+1}^m ; if $s_i \not\subseteq e_j$, $P_{i,j}$ is equal to the probability that s_i^n is supported by d_{j+1}^m . By splitting the problem in this way we can use the recursion in Lemma 2 to compute $P_{i,j}$ by means of the paradigm of dynamic programming.

Lemma 2

$$P_{i,j} = P(s_i \subseteq e_j) * P_{i+1,j+1} + P(s_i \not\subseteq e_j) * P_{i,j+1} \quad (4)$$

where $P(s_i \subseteq e_j)$ is the probability that s_i is contained in event e_j . And $P(s_i \subseteq e_j) = P(e_j \in d)$, if $s_i \subseteq e_j$; otherwise, $P(s_i \subseteq e_j) = 0$.

Proof. Referring to the law of total probability, we have:

$$P_{i,j} = P(s_i^n \preceq d_j^m | s_i \subseteq e_j) * P(s_i \subseteq e_j) + P(s_i^n \preceq d_j^m | s_i \not\subseteq e_j) * P(s_i \not\subseteq e_j)$$

where $P(s_i^n \preceq d_j^m | s_i \subseteq e_j) = P_{i+1,j+1}$ is the probability that $s' = \langle s_{i+1}, \dots, s_n \rangle$ is supported by sequence $\langle e_{j+1}, \dots, e_m \rangle$. And similarly we have $P(s_i^n \preceq d_j^m | s_i \not\subseteq e_j) = P_{i,j+1}$ is the support probability of s_i^n in d_{j+1}^m . \square

This dynamic schema is an adoption of the technique previously used in solving uncertain SPM [10] and frequent episode mining problems [11]. Using this dynamic programming scheme, we can compute the support probability by calculating the cells depicted in Fig.3. In the matrix, each cell relates to a probability $P_{i,j}$, with i marked on the x -axis and j marked on the y -axis. Referring to Lemma 2, we can compute $P_{i,j}$ from $P_{i,j+1}$ and $P_{i+1,j+1}$ which are cells to the right and lower right of $P_{i,j}$. By definition, if $s = \phi$, then $P(s \preceq d) = 1$; meanwhile, $P(s \preceq d) = 0$ if $s \neq \phi$ and $d = \phi$. Therefore, we iterate the cells from $P_{n+1,m+1}$ to $P_{1,1}$ so that we finally obtain $P(s \preceq d) = P_{1,1}$. The time complexity is $O(n * m)$, as we only need to iterate each cell once.

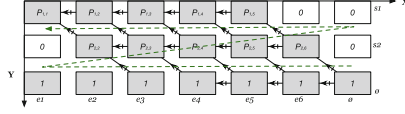


Fig. 3. An example of dynamic programming process

4.2 Distribute Dynamic Programming Schema

A direct extension of the dynamic programming approach in Spark needs to build a $n * m$ matrix for every support probability computation, and this might slow down the entire process because of expensive garbage collection overhead in Spark. Therefore, we refine the original DP schema and design a memory-efficient distributed dynamic programming (ddp) approach here. We first define $P_{s,j}$ as follows.

Definition 8. Given a sequential pattern s and an uncertain sequence $d = \langle e_1, \dots, e_m \rangle$, $P_{s,j}$ is defined to be the support probability $P(s \preceq d_j^m)$ where $d_j^m = \langle e_j, \dots, e_m \rangle$ is a subsequence of d .

The idea is that we save and reuse computational results in the last iteration. Based on the extension type of sequential pattern s , we have different dynamic programming schemas.

Sequence-Extended. If $s = \{i\} + s'$ is sequence-extended from another pattern s' , then we can compute the values of $P_{s,j}$ from $P_{s',j+1}$ and $P(s_1 \subseteq e_j)$ by Eq. (5), according to Lemma 2.

$$P_{s,j} = P(s_1 \not\subseteq e_j) * P_{s,j+1} + P(s_1 \subseteq e_j) * P_{s',j+1} \quad (5)$$

where $s_1 = \{i\}$ is the first element of s .

Item-Extended. Let $s = \{i\} \cup s'$, then s'_1 is a strict subset of s_1 and we have

$$P(s_1 \subseteq e_j) = \begin{cases} P(s'_1 \subseteq e_j) & \text{if } i \in e_j \\ 0 & \text{otherwise} \end{cases} \quad (6)$$

Referring to Lemma (2), we can compute $P_{s,j}$ by Eq. (7).

$$P_{s,j} = \begin{cases} P_{s,j+1}, & \text{if } i \notin e_j \\ P(s_1 \not\subseteq e_j) * P_{s,j+1} + P(s'_1 \subseteq e_j) * P(s_2^n \preceq d_j^m) & \text{otherwise} \end{cases} \quad (7)$$

Note that $s_2^n = s'^m_2 = \langle s_2, \dots, s_n \rangle$, then $P_{s',j}$ can be computed by:

$$\begin{aligned} P_{s',j} &= P(s'_1 \not\subseteq e_j) * P_{s',j+1} + P(s'_1 \subseteq e_j) * P(s'^m_2 \preceq d_j^m) \\ &= P(s'_1 \not\subseteq e_j) * P_{s',j+1} + P(s'_1 \subseteq e_j) * P(s_2^n \preceq d_j^m) \end{aligned} \quad (8)$$

ALGORITHM 1. ddpUpdate

Input: L_1 : a list of n non-zero $P(s_1'' \subseteq e_i)$ values ordered by eid i
 L_2 : a list of m non-zero $P_{s',j}$ values ordered by eid j
 $L_s \leftarrow \phi$, $p \leftarrow (n - 1)$, $q \leftarrow (m - 1)$
 $P_{s',j+1} \leftarrow 0$
while $p \geq 0$ **do**
 $P(s_1 \subseteq e_i) = P(s'' \subseteq e_i) \leftarrow L_1[p]$; // at event e_i
 $P_{s',j} \leftarrow L_2[q]$; // at event e_j
 while $j < i \wedge q \geq 0$ **do**
 /* find the nearest event e_j with non-zero value $P_{s',j}$ */
 $P_{s',i+1} = P_{s',j} \leftarrow L_2[q]$
 $q \leftarrow q - 1$
 end
 $q \leftarrow q + 1$
 if $L_2[q - 1] = P_{s',i}$ **then** $P_{s',i} \leftarrow L_2[q - 1]$;
 else $P_{s',i} \leftarrow L_2[q]$;
 compute $P_{s,i}$ by Equation (5) or (9) and insert it to the head of L_s
 $p \leftarrow p - 1$
end
return L_s

Therefore, we can derive Eq. (9) by substituting Eq. (8) into Eq. (7).

$$P_{s,j} = \begin{cases} P_{s,j+1} & \text{if } i \notin e_j \\ P(s_1 \not\subseteq e_j) * (P_{s,j+1} - P_{s',j+1}) + P_{s',j} & \text{otherwise} \end{cases} \quad (9)$$

Now we are able to compute $P_{s,j}$ from only the values of $P_{s',j}$, $P_{s',j+1}$ and $P(s_1 \subseteq e_j)$.

Eqs. (5) and (9) constitute our distributed dynamic programming structure for computing support probabilities in parallel. And the time complexity is $O(|d|)$.

4.3 Memory-Efficient Distributed SPM Algorithm

Lemma 3. *It is not necessary to save the value of $P_{s,j}$, if $P(s_1 \subseteq e_j) = 0$.*

Proof. Referring to Eqs.(5) and (9), if $P(s_1 \subseteq e_j) = 0$, then $P_{s,j} = P_{s,k}$ where e_k is the nearest event of e_j which has $k > j$ and $P(s_1 \subseteq e_k) > 0$. \square

Let s be a k -length sequential pattern generated by joining two $(k-1)$ -length patterns as $s = s'' \bowtie s'$. Then, the first element of s and s'' are identical when $k \geq 2$. For example, let $s = \langle (a)(bc)(d) \rangle$ and $s = s'' \bowtie s'$, then $s'' = \langle (a)(bc) \rangle$, $s' = \langle (bc)(d) \rangle$ so that $s_1 = s_1'' = (a)$.

Suppose L_1 is a list of non-zero $P(s_1'' \subseteq e_i)$ values and L_2 is a list of $P_{s',j}$ values with $P(s_1' \subseteq e_j) > 0$. Algorithm 1 computes the values of $P_{s,i}$ from L_1 and L_2 . For each value of $P(s_1'' \subseteq e_i) > 0$, we have $P(s \subseteq e_i) = P(s_1'' \subseteq e_i)$ at

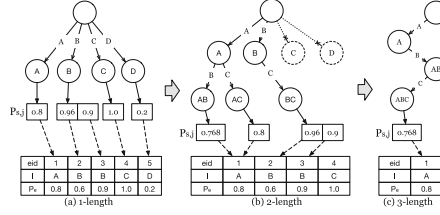


Fig. 4. An example process of updating candidate support probabilities

event e_i because $s_1 = s'_1$. Then we search the nearest event e_k , which satisfies $P_{s',k} > 0$ and $k > i$, to the right of e_i . Thus, we have $P_{s',i+1} = P_{s',k}$ by Lemma 3. If $P(s' \subseteq e_i) > 0$, the value of $P_{s',i}$ must have been saved and we can directly read it from L_2 ; if $P(s' \subseteq e_i) > 0$, $P_{s',i} = P_{s',i+1}$. Now that we have the values of $P(s \subseteq e_i)$, $P_{s',i+1}$ and $P_{s',i}$, we can compute $P_{s,i}$ by either Eqs. (5) or (9). The time complexity of Algorithm 1 is linear because both L_1 and L_2 are iterated only once.

We extend the data structure *prefix-tree* to save intermediate results in each iteration of uncertain SPM. The root of the prefix tree is the empty pattern ϕ . Each edge in the tree is associated with an item. The key of a node is identified by the path from root to that node. Values are not associated with inner nodes; only leaf nodes point to a list of $P_{s,j}$ values. Each value of $P_{s,j}$ is linked to the event e_j where $P(s_1 \subseteq e_j) > 0$. Here s_1 is the first element of s .

Figure 4 shows an example process of computing support probabilities with the new data structure. In Fig. 4(a), the leaf nodes are 1-length sequential patterns that are linked to events in an uncertain sequence. For example, $s = \langle B \rangle$ is associated with two values ($P_{s,2} = 0.96$, $P_{s,3} = 0.9$) which point to uncertain event e_2 and e_3 . The support probability of $\langle B \rangle$ is $P(s \preceq d) = P_{s,2}$ because e_2 is the first event where we have $P(s_1 \subseteq e_2) > 0$.

Another benefit of our algorithm is that we do not need to generate candidates in a centralized node; instead, we broadcast p-FSPs to all mappers and then generate candidates in parallel. In a map function applied to a sequence d , a p-FSP s is not participated in candidate generation if it is not supported in d . In Fig. 4, suppose S_1 is a set of 1-length p-FSPs and $\langle D \rangle \notin S_1$, then we can prune node D from the prefix tree and generate candidates by joining only three 1-length p-FSPs: $\langle A \rangle$, $\langle B \rangle$ and $\langle C \rangle$ for this sequence.

Consider the example candidate $s = \langle (A)(B) \rangle$. We first search leaf nodes associated with $s'' = \langle A \rangle$ and $s' = \langle B \rangle$ in the 1-length pattern tree. Then we retrieve $P(s''_1 \subseteq e_1) = 0.8$, $P_{s',2} = 0.96$ and $P_{s',3} = 0.9$ to compute $P_{s,1} = 0.8 * 0.96 = 0.768$ by Eq. (5). Thereafter, we generate a new leaf node $\langle AB \rangle$ for the 2-length prefix tree.

After expanding the tree for every possible 2-length candidate, we eliminate all 1-length patterns that have not been extended. For instance, node with $\langle C \rangle$ is prune because no 2-length candidate starting with item C are potentially supported in the sequence.

5 Evaluation

We implement our algorithms in Spark and evaluate the performance in large scale datasets. The uncertain SPM algorithm which directly adopts dynamic programming in Sect. 4.1 is denote by *basic*. We denote our distributed uncertain SPM algorithm in Sect. 4.3 by *dsp*. We also implement the IMRSPM algorithm [6] in Spark and name it as *uspm* here.

We employ the IBM market-basket data generator [3] to generate sequence datasets in different scales by varying the parameters: (1) C : number of sequences; (2) T : average number of events per sequence; (3) L : average number of items per event per sequence; (4) I : number of different items.

An existential probability α is added to each event in the synthetic datasets, where $\alpha \in [0.5, 0.9]$ is a parameter to control uncertain levels. We name a synthetic uncertain dataset by its parameters. For example, a dataset C10kT4L10I10k indicates $C = 10k = 10 * 1000$, $T = 4$, $L = 10$ and $I = 10k = 10 * 1000$.

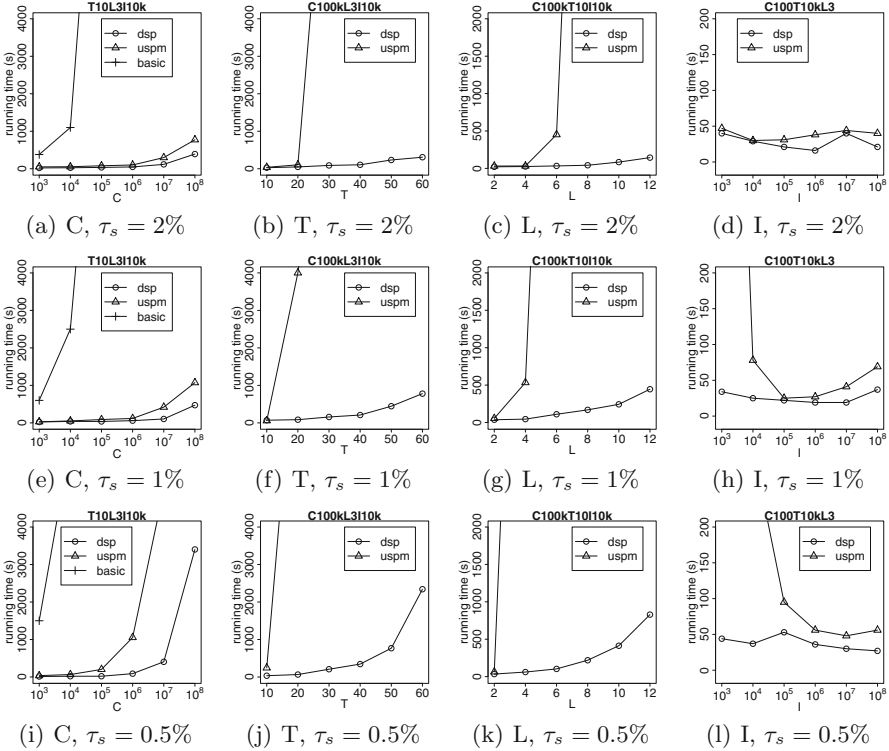


Fig. 5. Scalability of DSP algorithm

In Fig. 5, we evaluate the scalability of our DSP algorithm on various synthetic datasets in a Spark cluster with 100 nodes. Initially, we set uncertain level $\alpha = 0.8$ and frequentness probability threshold $\tau_p = 0.7$. In Fig. 5(a)–(d), we set $\tau_s = 2\% * C$; in Fig. 5(e)–(h), we have $\tau_s = 1\% * C$; and $\tau_s = 0.5\% * C$ in Fig. 5(i)–(l). Under each setting of τ_s , we vary the values of C , T , L and I to evaluate the performance of DSP in different scales:

- (1) Figure 5(a), (e), (i) show the running time variations of DSP when C varies from 1000 to 100 000 000, where $T = 10$, $L = 3$ and $I = 10k$.
- (2) Figure 5(b), (f), (j) show the running time variations of DSP when T varies from 10 to 60, where $C = 100k$, $L = 4$, $I = 10k$.
- (3) Figure 5(c), (g), (k) show the running time variations when L varies from 2 to 12, where $C = 100k$, $T = 4$, $I = 10k$.
- (4) Figure 5(d), (h), (l) show the running time variations when I varies from 10 000 to 10 000 000, where $C = 100k$, $T = 4$, $L = 4$.

We observe the following phenomenons in Fig. 5: (1) *dsp* outperforms *basic* and *uspm* under every setting of the parameters. Specifically, *dsp* is orders of magnitude faster than *basic* and *uspm* in datasets with larger values of T or L . When $C > 10k$, *basic* cannot finish because of the garbage collection overhead from Spark. This indicates that directly extending dynamic programming to Spark is not workable and also proves the advantage of our refined schemas. (2) The running time increase with the increment of C , T , L . This is intuitive because increasing these parameters generates larger scale datasets. Comparing to the C scale, both *dsp* and *uspm* are more sensitive to the increment of T and L . *uspm* fails quickly when T or L becomes larger; however, *dsp* still performs well even with large T or L values. (3) The running time first drops and then arises with the increment of I . When the value of I grows, the item occurrences become more sparse, and fewer p-FSPs are mined under the same thresholds; meanwhile, the volume of data shuffled from mapper to reducer via network increases because less key-value pairs are able to be pre-aggregated locally, which slows down the process when I is extremely large.

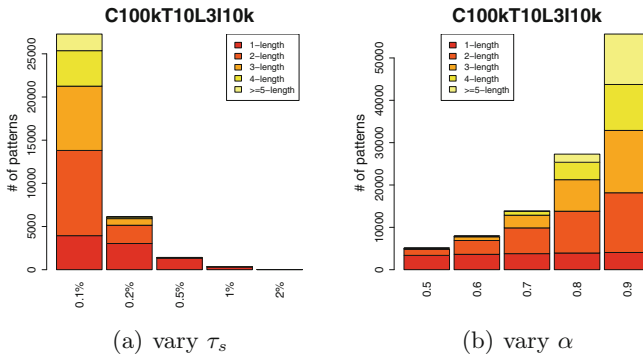


Fig. 6. Number of p-FSPs with different τ_s and α settings

Figure 6(a) shows the number of p-FSPs in the dataset C100kT10L3I10k with uncertain level $\alpha = 0.8$, where we vary the value of τ_s from 0.1% to 2%; Fig. 6(b) shows the effect of uncertain level α to the number of p-FSPs in C100kT10L3I10k, where $\tau_s = 0.1\%$ and $\alpha \in [0.5, 0.9]$. From Fig. 6, we observe that: (1) With the increment of τ_s , the number of p-FSPs decreases dramatically. This is intuitive because a larger minimal support threshold makes fewer candidates be probabilistically frequent. (2) With the increment of α , the number of p-FSPs increases, which shows the effect of uncertainty in SPM problems. When uncertain level is high (α is small), there are fewer precise information in the data, which makes it more difficult to find p-FSPs under the same thresholds.

6 Conclusions

In this paper, we design a distributed dynamic programming method in Spark to mine sequential patterns in large scale uncertain databases. Our algorithm is proved to be efficient and highly scalable. In the future, we will continue to work on integrating constraints in large scale uncertain SPM problems.

References

1. Agarwal, A., Xie, B., Vovsha, I., Rambow, O., Passonneau, R.: Sentiment analysis of twitter data. In: Proceedings of the Workshop on Languages in Social Media, pp. 30–38 (2011)
2. Aggarwal, C.C., Yu, P.S.: A survey of uncertain data algorithms and applications. *IEEE Trans. Knowl. Data Eng.* **21**(5), 609–623 (2009)
3. Agrawal, R., Srikant, R.: Fast algorithms for mining association rules in large databases. In: VLDB, pp. 487–499 (1994)
4. Bernecker, T., Kriegel, H.P., Renz, M., Verhein, F., Zuefle, A.: Probabilistic frequent itemset mining in uncertain databases. In: SIGKDD, pp. 119–128. ACM (2009)
5. Chen, C.C., Tseng, C.Y., Chen, M.S.: Highly scalable sequential pattern mining based on mapreduce model on the cloud. In: BigData Congress, pp. 310–317 (2013)
6. Gao, Y., Sun, Z., Wang, Y., Liu, X., Yan, J., Zeng, J.: A comparative study on parallel LDA algorithms in mapreduce framework. In: Cao, T., Lim, E.P., Zhou, Z.H., Ho, T.B., Cheung, David, Motoda, Hiroshi (eds.) PAKDD 2015. LNCS, vol. 9078, pp. 675–689. Springer, Heidelberg (2015)
7. Jestes, J., Cormode, G., Li, F., Yi, K.: Semantics of ranking queries for probabilistic data. *IEEE Trans. Knowl. Data Eng.* **23**(12), 1903–1917 (2011)
8. Li, Y., Bailey, J., Kulik, L., Pei, J.: Mining probabilistic frequent spatio-temporal sequential patterns with gap constraints from uncertain databases. In: IEEE International Conference on Data Mining, pp. 448–457 (2013)
9. Miliaraki, I., Berberich, K., Gemulla, R., Zoupanos, S.: Mind the gap: large-scale frequent sequence mining. In: SIGKDD, pp. 797–808 (2013)
10. Muzammal, M., Raman, R.: Mining sequential patterns from probabilistic databases. In: PAKDD, pp. 210–221 (2011)
11. Wan, L., Chen, L., Zhang, C.: Mining frequent serial episodes over uncertain sequence data. In: EDBT, pp. 215–226 (2013)

12. Zaharia, M., Chowdhury, M., Das, T., Dave, A., Ma, J., McCauley, M., Franklin, M.J., Shenker, S., Stoica, I.: Resilient distributed datasets: a fault-tolerant abstraction for in-memory cluster computing. In: NSDI 2012 (2012)
13. Zhao, Z., Yan, D., Ng, W.: Mining probabilistically frequent sequential patterns in uncertain databases. In: EDBT, pp. 74–85 (2012)
14. Zhao, Z., Yan, D., Ng, W.: Mining probabilistically frequent sequential patterns in large uncertain databases. *IEEE Trans. Knowl. Data Eng.* **26**, 1171–1184 (2013)

Advances in Knowledge Discovery and Data Mining
20th Pacific-Asia Conference, PAKDD 2016, Auckland,
New Zealand, April 19-22, 2016, Proceedings, Part II
Bailey, J.; Khan, L.; Washio, T.; Dobbie, G.; Huang, J.Z.;
Wang, R. (Eds.)
2016, XXIV, 572 p. 156 illus., Softcover
ISBN: 978-3-319-31749-6