Mining Sequential Patterns from Probabilistic Data

Thesis submitted for the degree of
Doctor of Philosophy
at the University of Leicester

by

Muhammad Muzammal
Department of Computer Science
University of Leicester

September 2012

Declaration of Authorship

I hereby declare that content of this thesis is my own work and that it is the result of work done during the period of registration. To the best of my knowledge, it contains no material previously published or written by another person nor material which to a substantial extent has been accepted for the award of any other degree or diploma of the university or other institute of higher learning, except where due acknowledgement has been made in the text.

Parts of this thesis appeared in the following publications, to each of which I have made substantial contributions:

- Muhammad Muzammal and Rajeev Raman. Uncertainty in Sequential Pattern Mining. *In Proceedings of 27th British National Conference on Databases (BNCOD)*, volume 6121 of *Lecture Notes in Computer Science*, pages 147-150, Springer-Verlag, 2010.

- Muhammad Muzammal and Rajeev Raman. On Probabilistic Models for Uncertain Sequential Pattern Mining. *In Proceedings of 6th International Conference on Advanced Data Mining and Applications (ADMA)*, volume 6440 of *Lecture Notes in Computer Science*, pages 60-72, Springer-Verlag, 2010.

- Muhammad Muzammal and Rajeev Raman. Mining Sequential Patterns from Probabilistic Databases. *In Proceedings of 15th Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD)*, volume 6635 of *Lecture Notes in Computer Science*, pages 210-221, Springer-Verlag, 2011.

- Muhammad Muzammal. Mining Sequential Patterns from Probabilistic Databases by Pattern-Growth. *In Proceedings of 28th British National Conference on Databases (BNCOD)*, volume 7051 of *Lecture Notes in Computer Science*, pages 118-127, Springer-Verlag, 2011.

Some of the technical contributions in this thesis have not yet been published and are listed below:

- The definition of the SLU-D model in Section 4.1.3 and the complexity results for evaluating the interestingness predicate for the SLU-D model in Section 5.1.3.
- The evaluation of the effectiveness of the probabilistic SPM framework in the presence of noise in Section 7.4.

Abstract

Sequential Pattern Mining (SPM) is an important data mining problem. Although it is assumed in classical SPM that the data to be mined is deterministic, it is now recognized that data obtained from a wide variety of data sources is inherently noisy or uncertain, such as data from sensors or data being collected from the web from different (potentially conflicting) data sources. Probabilistic databases is a popular framework for modelling uncertainty. Recently, several data mining and ranking problems have been studied in probabilistic databases. To the best of our knowledge, this is the first systematic study of mining sequential patterns from probabilistic databases.

In this work, we consider the kind of uncertainties that could arise in SPM. We propose four novel uncertainty models for SPM, namely tuple-level uncertainty, event-level uncertainty, source-level uncertainty and source-level uncertainty in deduplication, all of which fit into the probabilistic databases framework, and motivate them using potential real-life scenarios. We then define the interestingness predicate for two measures of interestingness, namely expected support and probabilistic frequentness. Next, we consider the computational complexity of evaluating the interestingness predicate, for various combinations of uncertainty models and interestingness measures, and show that different combinations have very different outcomes from a complexity theoretic viewpoint: whilst some cases are computationally tractable, we show other cases to be computationally intractable.

We give a dynamic programming algorithm to compute the source support probability and hence the expected support of a sequence in a source-level uncertain database. We then propose optimizations to speedup the support computation task. Next, we propose probabilistic SPM algorithms based on the candidate generation and pattern growth frameworks for the source-level uncertainty model and the expected support measure. We implement these algorithms and give an empirical evaluation of the probabilistic SPM algorithms and show the scalability of these algorithms under different parameter settings using both real and synthetic datasets. Finally, we demonstrate the effectiveness of the probabilistic SPM framework at extracting meaningful patterns in the presence of noise.

In the name of Allah, the Beneficent, the Merciful.

Acknowledgements

First and foremost, I wish to thank Prof. Rajeev Raman for playing a pivotal role in this research. To me Rajeev was both a supervisor and a teammate as he encouraged and supported me on many levels. I am grateful to him for all the labour and the pains that he took in training me; without him this thesis will not exist.

Bahria University, Pakistan and Higher Education Commission, Pakistan were the funding bodies and are to be thanked for supporting this research.

I would also like to thank the staff and my fellow PhD students at the Department of Computer Science for providing a conducive research environment.

I express my special gratitude to my examiners, Dr. Frans Coenen and Dr. Neil Walkinshaw, for the useful recommendations in making this thesis a more comprehensible document.

Finally, a big humongous thanks to my parents, my brothers and sister for their all-around support especially during this time, and I am sure that my uncle (late) would have been very proud seeing me getting this far.

Thank you!

Muzammal

Contents

List of Figures

List of Tables

List of Algorithms

Abbreviations

BFS	**B**readth **F**irst **S**earch
DFS	**D**epth **F**irst **S**earch
DP	**D**ynamic **P**rogramming
ELU	**E**vent **L**evel **U**ncertainty
GSP	**G**eneralized **S**equential **P**atterns
KDD	**K**nowledge **D**iscovery in **D**atabases
PGA	**P**attern **G**rowth **A**pproach
SLU	**S**ource **L**evel **U**ncertainty
SLU-D	**S**ource **L**evel **U**ncertainty in **D**eduplication
SPADE	**S**equential **PA**ttern **D**iscovery using **E**quivalence classes
SPAM	**S**equential **PA**ttern **M**ining
SPM	**S**equential **P**attern **M**ining
TLU	**T**uple **L**evel **U**ncertainty

Dedicated to my parents.

Chapter 1

Introduction

Knowledge Discovery in Databases (KDD) is an important research area at the intersection of databases, machine learning and artificial intelligence. Fayyad et al. [5] define KDD as "the non-trivial process of identifying valid, novel, potentially useful, and ultimately understandable patterns in data". Data mining, the analysis step of KDD, is aimed at analysing huge volumes of data by automated (or semi-automated) means in order to discover useful patterns, interesting events or trends. It should also be noted that data mining is not limited to traditional relational or transaction databases, but can be applied to a variety of data such as stream data, sensor data, text databases, time-series or sequence data. Fayyad et al. [6] classify the data mining algorithms into two groups:

1. *Predictive* algorithms which predict an unknown attribute value based on some known attribute values in data, e.g. classification, deviation detection, etc.

2. *Descriptive* algorithms which discover underlying patterns or correlations in data that describe the data. Clustering, association rule mining and sequential pattern mining are examples of descriptive data mining tasks.

We now briefly give an overview of the algorithms one each from the above classes of algorithms, i.e. we describe classification (a predictive algorithm) and association rule mining (a descriptive algorithm). See Han et al. [7], for a good introduction to data mining concepts and techniques.

Classification

Classification is the task of predicting the class of an object from a pre-defined set of classes by examining its attributes in a systematic way. To perform classification, the input dataset is split into a training and a test set. A record in the input dataset contains a set of attribute, one of which is the class attribute. The classification algorithm is trained on the training data so as to be able to predict the value of the class attribute for an unseen record. The objective of classification is that the class label for a new record should be as accurate as possible. Once a classification model has been built, the test set is used to validate the model.

Classification has been applied to a variety of application domains.

- An important application of classification is in direct marketing. In the mobile phone industry, a company aims at reducing the posting costs by targeting a group of customers that are likely to sign a contract for a specific new mobile phone. When a customer (new or existing) signs up for a mobile phone contract, different kinds of information is solicited about the customer, e.g. gender, age, marital status, job, salary range, etc. The history data for a similar product can be used to classify customers as those who will sign up or otherwise, for a new mobile phone.

- Classification has been used for sky survey cataloguing. The objective is to classify a sky object as a star or a galaxy, based on the extremely high definition telescopic survey images (23040 $\times$ 23040 pixels per image) available from

Palomar Observatory. As reported in [8], the classification algorithms were able to find 16 new high red-shift quasars (a quasar is a young or an emerging galaxy and a redshift is a measure to represent far off distances, e.g. millions of light years) from the Palomar Observatory images.

Association Rule Mining

Association rule mining is the task of discovering associations among itemsets (sets of items) in a transaction database where each transaction is given as a set of items. Association rules have also been studied in many application domains.

- An important application domain of association rule mining is in the retail environment. In the retail environment, association rules can be used in marketing, sales promotions, shelf management, etc. For example, to answer questions like which items sell together, process the market basket data recorded by barcode scanners for each basket, in order to find associations among items. For example, an association rule of the form $\{\text{bagels}\} \implies \{\text{tortilla chips}\}$ might suggest that if a customer buys bagels, then he is quite likely to buy tortilla chips. These two items can be placed next to each other in order to improve shelf management.

- Another application of association rule mining is in inventory management. For example, a consumer appliance company wants to anticipate the nature of the repair requests on its consumer products. The idea is to equip the service vehicles with the 'right' kind of tools and parts in order to reduce visits to the consumer household. This can be achieved by processing the data on tools and parts that were needed in repairs in different regions.

An important concern in association rule mining, which we do not pursue in this thesis, is how to find rules that are useful or in other words, how to avoid redundant or uninteresting rules. For example, it has been noted that only few of the reported association rules are actionable, whereas most of them are either trivial (obvious to the people in the business domain) or inexplicable (can not be explained and are not actionable). Many alternative formulations have been proposed in literature for mining interesting association rules, e.g. [9, 10] etc.

Initial studies on association rule mining focus on discovering intra-transaction rules only, i.e. associations among the items in a transaction [11]. Agrawal and Srikant [12] introduced the idea of sequential patterns, a variation of association rules that aims at finding inter-transaction association rules in a temporal database, i.e. finding associations within as well as between transactions ordered by a time-stamp.

In this work, we focus on association rule mining in a temporal database or more specifically, Sequential Pattern Mining (SPM).

1.1 Sequential Pattern Mining

SPM is a well-researched area in data mining [13–15]. The given input is a database which is a set of records, where each record has a (unique) time-stamp, an event attribute (which is a set of items) and a source attribute (that caused the event); the objective is to find all frequent sequential patterns in data. Sequential patterns have been studied in many application domains, e.g. in market basket analysis, web usage mining, genome and sequence data analysis, etc. We briefly review a few of them here.

- Sequential patterns have been used for web personalization [16]. For example, a click stream of page visits by a user can be seen as a sequence of events by

that user. Then, frequent sequential patterns are extracted in order to find the frequent user navigation paths. Sequential patterns can then be used to predict the next page to be visited (based on the sequence of page visits by the user so far) and predictive tasks such as web pre-fetching can be performed in order to improve the navigation experience of the user.

- In text categorization, a collection of text documents is transformed to a sequence database where each document corresponds to a sequence, and each sentence in a document is considered as an event, and the order of the sentences is the order in which they appear in the document. Then, the frequent sequential patterns are discovered which are used to categorize documents [17].

- Sequential patterns have also been used in protein fold recognition [18]. The task is to assign a protein sequence whose structure is unknown, to one of the folds (a fold is 3D structure of a protein) in order to determine the protein function. Frequent sequential patterns are extracted from all folds and then each of the extracted sequential patterns is checked to see if it is contained in the protein. The protein is assigned to the fold for which it contains the most sequential patterns.

1.2 Uncertain Data

Data to be mined that is obtained from real-life applications is often incomplete/incorrect/imprecise/inconsistent due to various reasons (a general term for these kinds of errors is that the data is "dirty" [2]), for example:

(a) data may be incomplete, as the values for some attributes of interest can be missing; can happen due to data entry errors or values are unknown/unobtainable at the time of data collection.

(b) data may contain noise, for instance salary = "-20000".

(c) data may be imprecise; can happen in situations where an attribute value is based on a range of values, e.g. age = [25,34].

(d) data may contain inconsistencies, for example in deduplication [2], i.e. when multiple tuples in a database (potentially) represent the same real world entity.

Classical association rule mining works under the assumption that the data to be mined is clean or entirely determined, as it is assumed that all the noise or inconsistencies in data have been eliminated in the Data Preprocessing, or more specifically, Data Cleansing phase [7, Chapter 2]. However, data cleansing may result in valuable information loss [2].

Recently, many new applications have emerged which generate data that is inherently noisy or uncertain [19]. For example, sensor or RFID data is uncertain due to reasons like equipment limitations or sampling based data collection [20]. Similarly, uncertainty is introduced in data when data is being extracted from multiple (potentially conflicting) sources on the web [21]. In applications like tracking a mobile object, the trajectory of the object can be predicted only with a degree of confidence [22]. In some applications, uncertainty has to be introduced, e.g. due to the increasing importance of privacy concerns [23]. In deduplication, a new approach is to leave some uncertainty in data rather than complete data cleansing [2].

An important point about all these and other applications of uncertain data is that it is somehow possible to quantify the level of uncertainty. For example, in deduplication, probabilities are assigned to potential duplicates and tuples which are more likely to be 'true' are assigned higher probabilities. Trio [21] (a probabilistic database management system) computes the confidence in a tuple or an attribute value by looking at the source of information. It should also be noted that the above

kind of uncertainties arise either in the tuple (tuples have existential probabilities) or in an attribute (an attribute value is uncertain having a small number of precise alternatives along with the associated probabilities while the rest of the attribute values are certain).

This thesis examines issues that arise in SPM when considering that data to be mined can be uncertain:

(1) How should different kinds of uncertainties in data for SPM be modelled formally?

(2) How should the frequentness criteria be defined for SPM from uncertain data?

(3) Is it possible to develop scalable and high performance algorithms for SPM from uncertain data?

(4) How effective is the probabilistic SPM framework ((1)—(3) above is the probabilistic SPM framework) in extracting meaningful patterns from uncertain data.

Although we consider this in detail later in the thesis, we now briefly introduce an approach for (1). *Probabilistic databases* is a popular framework for modelling (such kind of) uncertainties in data [24]. A probabilistic database is a set of *possible worlds* where each world is consistent with a given schema and is the true world only with some probability. The set of possible worlds is considered the complete model as it can capture any kind of correlations or dependencies in data. However, there can be an exponential number of possible worlds and it may not be feasible to enumerate the complete set of possible worlds. In the literature, some simplifying assumptions are made at the cost of expressiveness in order to model uncertain data, e.g. assuming independence among tuples.

1.3 Motivation

Recently, several data mining problems have been studied in the context of uncertain data [19]. There have been quite a few studies in the literature on association analysis [25] and more specifically, frequent itemset mining [26–29] from uncertain data. We focus on sequential pattern mining. We observe that uncertainty in SPM could arise in many real-life situations.

- Consider for example a scenario where employees' movements (sequence of activities) are tracked in a building using RFID sensors. Since an RFID antenna has only some probability of reading a tag within its range, the stream of tags read by RFID sensors is used to output an *uncertain* relation such as `MEETING(time, person1, person2, room, prob)`. An example tuple in `MEETING` could be (103, 'Alice', 'Bob', 435, 0.4), which means that at time 103, Alice and Bob are having a meeting (event) with probability 0.4 in room 435 (source) [20]; thus, the meeting event only has a probability of 0.4 of occurring, which shows tuple-level uncertainty.

- Consider a scenario where a logged-in user (source) enters (a sequence of) search terms into a search engine (events). The search terms could be disambiguated in many different ways (uncertainty in the event attribute). For example, a search term 'Tiger' could potentially be disambiguated as $\{(\text{Animal}, 0.4), (\text{Sports Personality}, 0.3), (\text{Insurance}, 0.2), \ldots\}$.

- A customer (source) purchases some items (event) from a superstore, and provides identity information, e.g. by filling a form. The same customer may fill a *new* form in a subsequent visit and thus, multiple matches may emerge in the customer database as the customer's details may be incomplete or incorrect (uncertainty in the source attribute).

To the best of our knowledge, there has been no attempt to study SPM in probabilistic databases, and this is the subject of this thesis.

We first give our main contributions and then give a thesis outline.

1.4 Our Contributions

We study the sequential pattern mining problem in probabilistic databases. Our main contributions are as follows:

(a) We consider the kind of uncertainties that could arise in SPM and model them in probabilistic databases framework either as tuple-level uncertainty or attribute-level uncertainty [24]. In tuple-level uncertainty, tuples have existential probabilities; and in attribute-level uncertainty, we consider uncertainty in the event or in the source attribute. We also consider another kind of uncertainty that could arise in the source attribute, motivated by deduplication. Thus, we consider four novel uncertainty models for SPM, namely tuple-level uncertainty, event-level uncertainty, source-level uncertainty and source-level uncertainty in deduplication, all of which fit into the probabilistic databases framework, and motivate them using potential real-life scenarios.

(b) We define an interestingness predicate [30] which tests if a sequence is frequent in the context of SPM in probabilistic databases, and define two measures of interestingness, namely expected support and probabilistic frequentness, based upon existing definitions for frequent itemset mining in probabilistic databases [28, 31].

(c) We discuss evaluating the interestingness predicate from a complexity theoretic viewpoint and show that different combinations of uncertainty models

and interestingness measures have very different outcomes from a complexity theoretic viewpoint. Thus, whilst some cases are computationally tractable, we show others to be computationally intractable.

(d) We give a dynamic programming based algorithm for computing the source support probability and hence, the expected support of a sequence in a source-level uncertain (SLU) database. Further, we propose optimizations to speed up the support computation task. Then, we extend classical SPM algorithms based on the candidate generation and pattern growth frameworks to work under probabilistic settings for the SLU model using the expected support measure. In all, we propose two candidate generation algorithms, one based on a breadth-first and one based on a depth-first exploration of the search space, as well as a pattern growth algorithm based on the idea of projected databases.

(e) We give an empirical evaluation of the optimizations and algorithms that we consider, and demonstrate the scalability of our algorithms under different parameter settings using both synthetic and real datasets. We also evaluate the effectiveness of the probabilistic SPM framework at extracting meaningful patterns in the presence of noise.

1.5 Thesis Organisation

The rest of the thesis is organized as follows.

- Chapter 2 gives an overview of SPM. We define the SPM problem, give an overview of some popular classical SPM algorithms, discuss some alternative SPM formulations that have been proposed in literature, and also discuss a few applications of SPM.

- Chapter 3 is a discussion on probabilistic databases. It gives an overview of probabilistic data models proposed in literature and the work on top-k and frequent itemset mining in the context of probabilistic databases.

- In Chapter 4, we describe the kind of uncertainties that could arise in SPM and give probabilistic data models to model such uncertainties. We then define interestingness predicates for probabilistic SPM.

- Chapter 5 presents a detailed discussion on the computational complexity of evaluating the interestingness predicate under different uncertainty models and measures of interestingness. We also give a dynamic programming based algorithm to compute the expected support of a sequential pattern in a source-level uncertain database.

- In Chapter 6, we consider the source-level uncertainty model and the expected support measure, and give probabilistic SPM algorithms based on both candidate generation and pattern growth frameworks to find the set of frequent sequential patterns.

- Chapter 7 presents an empirical evaluation of the algorithms proposed in Chapter 6. We demonstrate the scalability of our algorithms under different parameter settings. We also give an evaluation on the effectiveness of the probabilistic SPM framework in the presence of noise.

- Chapter 8 concludes this thesis and gives an outline of future work.

Chapter 2

Sequential Pattern Mining

Association rule mining was first proposed by Agrawal et al. [11], and is a well researched area in data mining [32, 33]. Agrawal and Srikant [12] introduced sequential patterns, which are association rules with a temporal order.

In this chapter, we first give an overview of association rule mining. Then, we define SPM and discuss the computational complexity of SPM as well. Next, we discuss some classical SPM algorithms and contrast their main characteristics. We also discuss some alternative SPM formulations. Finally, we give some applications of SPM.

2.1 Association Rule Mining

Association rule mining was first proposed in the context of market basket analysis [11]. Since its introduction, the problem has been applied to many application domains. See Han et al. [32] for a detailed overview.

The association rule mining problem can be decomposed into two sub-problems. First, *frequent itemset mining*, which involves finding all the frequent itemsets in the database, and second, *association rule discovery*, which involves finding the associations or correlations among discovered frequent itemsets. We elaborate on each of these below.

2.1.1 Frequent Itemset Mining

We formally define the frequent itemset mining problem. Let $\mathcal{I} = \{i_1, i_2, \ldots, i_q\}$ be a set of *items*. A set $U \subseteq \mathcal{I}$ is called an *itemset*. A *database* $D = \{r_1, r_2, \ldots, r_n\}$ is a set of *records* such that each record $r_i \in D$ is of the form (rid, I), where rid is a unique record identifier and $I \subseteq \mathcal{I}$ is a set of items. A record $r_i = (rid, I)$ *supports* an itemset U if $U \subseteq I$. For a record $r_i = (rid, I)$ and an itemset U, let $X_i(U, D)$ be an indicator variable whose value is 1 if r_i supports U and 0 otherwise. For an itemset U, define the *support* of U in D, denoted by $Sup(U, D)$, as follows:

$$Sup(U, D) = \sum_{i=1}^{n} X_i(U, D). \tag{2.1}$$

An itemset U is considered *frequent* if $Sup(U, D)$ is at least some user-defined support threshold θ, $0 \leq \theta \leq n$.

Definition 2.1 (**Frequent Itemset Mining Problem).** Given a database D having n records, and a user-defined support threshold θ, $0 \leq \theta \leq n$, find all frequent itemsets in D.

We illustrate the above concepts with the help of examples using the sample database D of Table 2.1. Each record in D is example of an itemset. The record with $rid = 2$ supports an itemset $\{b\ c\}$ (as $\{b\ c\} \subseteq \{b\ c\ e\}$). The support of an itemset $\{b\ c\}$ is 2 as records with $rid = 2$ and 3, support $\{b\ c\}$. Assuming a minimum support

TABLE 2.1: A sample database.

rid	*I*
1	a c d
2	b c e
3	a b c e
4	b e

threshold $\theta = 2$, an itemset $\{a\}$ is frequent (as $Sup(\{a\}, D) = 2$) whereas $\{d\}$ is not (as $Sup(\{d\}, D) = 1$).

2.1.2 Association Rule Discovery

An association rule is an implication of the form $A \Rightarrow B$ where $A, B \subseteq \mathcal{I}$, $A \cap B = \emptyset$. The rule is read as A implies B which means that if a record supports an itemset A, it will also support the itemset B. The *confidence* in an association rule $A \Rightarrow B$, denoted by $Conf(A \Rightarrow B)$, is defined as follows:

$$Conf(A \Rightarrow B) = \frac{Sup(A \cup B, D)}{Sup(A, D)}.$$

For example, if $Conf(A \Rightarrow B) = 70\%$, it means that if a record supports A, it also supports B, at least 70% of the times. We now formally define the association rule mining problem:

Definition 2.2 (**Association Rule Mining Problem).** Given a database D having n records and two user-defined thresholds, a support threshold θ, $0 \leq \theta \leq n$, and a confidence threshold τ, $0 \leq \tau \leq 1$, find all association rules having support at least θ and confidence at least τ.

In the sample database of Table 2.1, $\{b\ c\}$ is a frequent itemset as $Sup(\{b\ c\}, D) = 2$, and the confidence in $b \Rightarrow c$ is 66.6% as for two out of three transactions in D where $\{b\}$ occurs, $\{c\}$ also occurs.

This concludes our discussion of association rule mining. We have defined the association rule mining problem and have illustrated some basic concepts with the help of examples. We now discuss sequential pattern mining.

2.2 Sequential Pattern Mining

The SPM problem was first proposed by Agrawal and Srikant [12], and has been studied extensively in literature (see [13–15] among many others). The problem was first proposed in the context of market basket data and has been studied in many application domains since then (see Section 2.6).

For example, in market basket data analysis [12], customers visit a superstore multiple times and purchase items in each visit. As the customers and the associated transactions are recorded by the store, it is possible to analyse the items (or sets of items) customers purchase in successive visits and thus the future purchasing behaviour of a customer can be predicted.

We now formally define the SPM problem.

2.2.1 Problem Statement

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 record $r_i \in D$ 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. In the above market basket data analysis example, a customer is considered as a source and the associated transactions are considered as events which are ordered by a time-stamp.

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 by $s \preceq t$, 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 by $Sup(s, D)$, as:

$$Sup(s, D) = \sum_{i=1}^{m} X_i(s, D). \tag{2.2}$$

A sequence s is considered frequent if $Sup(s, D)$ is at least some user-defined support threshold θ, $0 \leq \theta \leq m$. We now formally define the SPM problem:

Definition 2.3 (**SPM Problem).** Given a database D and a user-defined support threshold θ, $0 \leq \theta \leq m$, find all frequent sequences in D.

Thus in market basket data analysis, the objective is find the sequences of items (or itemsets) that are purchased by a considerable fraction of customers.

We illustrate the above concepts with the help of examples using the sample database D of Figure 2.1. Consider for example the sequence $s = \langle (a)(b, c) \rangle$. The sets (a) and (b, c) are elements of s, and the length of s is 3. The sequence s is a subsequence of D_X, the source sequence for source X (denoted as $\langle (a)(b, c) \rangle \preceq$

eid	e	σ
e_1	(a, b, d)	X
e_2	(b)	Y
e_3	(b, c, d)	X
e_4	(a, b, c)	Y
e_5	(a, b)	Z
e_6	(b, c, d)	X
e_7	(b, c, d)	Z

⇓

source	*sequence*
D_X	$(a, b, d)(b, c, d)(b, c, d)$
D_Y	$(b)(a, b, c)$
D_Z	$(a, b)(b, c, d)$

(Reproduced from Ayres et al. [34])

FIGURE 2.1: A sample database (top) transformed to source sequences (bottom).

$(a, b, d)(b, c, d)(b, c, d))$, and is not a subsequence of D_Y (denoted as $\langle(a)(b, c)\rangle \not\preceq (b)(a, b, c)$). Assuming a minimum support threshold $\theta = 2$, $\langle(a)(b, c)\rangle$ is frequent (as $Sup(\langle(a)(b, c)\rangle, D) = 2$) whereas $\langle(c)(d)\rangle$ is not (as $Sup(\langle(c)(d)\rangle, D) = 1$).

Classical SPM finds the complete set of frequent sequential patterns. However, the number of sequential pattern can be huge, as for any frequent k-sequence, all of its subsequences are also reported. In the literature, some compact definitions have been proposed [35]:

Definition 2.4 (**Maximal Sequence**). A frequent sequence s is maximal if there is no other frequent sequence s' such that $s \preceq s'$.

Definition 2.5 (**Closed Sequence**). A frequent sequence s is closed if there is no other frequent sequence s' such that (1) $s \preceq s'$ and (2) $Sup(s, D) = Sup(s', D)$.

Yan et al. [35] have shown that the set of frequent sequential patterns contains the set of closed sequential patterns, which in turn contains the set of maximal sequential patterns. The problem of maximal/closed SPM is defined analogously to SPM and has been studied in the literature [14].

We now briefly discuss the computational complexity of the SPM problem. For an overview of complexity classes, see Appendix A.

2.2.2 Computational Complexity of SPM

The input to the SPM problem is a database D and a support threshold θ, and the output is the set of frequent sequential patterns. We focus on some of the fundamental aspects of SPM, and consider the computational complexity of the following problem:

Problem 2.6. *Given a database D, a sequential pattern s and a support threshold θ, determine whether s is frequent or not.*

We can compute the support of s in D in a single scan of the database D and thus, to check if a sequential pattern is frequent is in polynomial time. In the framework described by Gunopulos et al. [30], the above problem is the *interestingness* or *quality* predicate and all the SPM algorithms in essence must evaluate the interestingness predicate.

Although the interestingness predicate for a single sequence can be evaluated in polynomial time, this does not mean that finding all frequent sequential patterns can be done in polynomial time. Zaki [36] have shown that given an arbitrary support threshold θ and a database D having n frequent items, the number of frequent sequences upto length k could at most be $O(n^k)$ and thus, the number of frequent sequential patterns need not be polynomial in the size of the input database.

As already mentioned, the set of frequent sequential patterns contains the set of maximal sequential patterns. Some hardness results have been shown in the literature for mining maximal sequential patterns and thus are true for frequent sequential patterns as well. Gunopulos et al. [30] have shown that given an arbitrary support threshold θ, it is NP-complete to decide if there is a maximal sequential pattern with at least k items. It is therefore, NP-hard to enumerate the set of maximal sequential patterns with at least k items. Further, Yang [37] have shown that the problem of counting the number of maximal sequential patterns is #P-complete.

In summary, although the support of a sequential pattern (i.e. the interestingness predicate for SPM) can be computed in polynomial time, the set of frequent sequential patterns or maximal sequential patterns can be exponentially larger than the size of the input database. Thus, a reasonable support threshold is usually assumed in SPM algorithms to restrict the number of reported sequential patterns.

We now give an overview of some classical SPM algorithms.

2.3 SPM Algorithms

Many SPM algorithms have been proposed in the literature. These can broadly be classified into two classes, those based on the candidate generation framework and those based on the pattern growth framework. SPM algorithms are different in:

- *Search space exploration:* that is, the way sequential patterns are discovered.
- *Support computation:* that is, the way sequential patterns are determined to be frequent or otherwise.

We now review some SPM algorithms based on both the candidate generation and the pattern growth framework.

2.3.1 Candidate Generation

The candidate generation algorithms are all based on the apriori property which is an anti-monotonic property and is stated as follows:

Definition 2.7 (**Apriori Property).** For any two sequences s and s', if $s \preceq s'$ then $Sup(s, D) \geq Sup(s', D)$.

The apriori property means that if a sequence s is not frequent, any of the supersequences of s will also not be frequent.

We review some of the popular candidate generation algorithms from literature, namely GSP (**G**eneralized **S**equential **P**atterns) [38], SPADE (**S**equential **PA**ttern **D**iscovery using **E**quivalence classes) [36] and SPAM (**S**equential **PA**ttern **M**ining) [34].

In our examples, we consider the sample database of Figure 2.1 and assume that the support threshold $\theta = 3$. We denote the set of candidate j-sequences by C_j and the set of frequent j-sequences by L_j.

2.3.1.1 GSP

The GSP algorithm [38] assumes that the input database is in source sequence format and it makes multiple passes over the database to find the set of sequential patterns. In the first pass, the set of frequent 1-sequences, denoted by L_1, is discovered. For example, for the sample database of Figure 2.1, support computation is performed for all 1-sequences $\{(a) : 3, (b) : 3, (c) : 3, (d) : 2\}$ and thus, $L_1 = \{(a), (b), (c)\}$ is obtained. For $j = 2$ onwards, L_{j-1} is used to generate the *set of candidate j-sequences* denoted by C_j. Next, L_1 is used to generate C_2 and subsequently, L_2 is obtained after support computation. For example, C_2 is generated from L_1, and after support computation, $C_2 = \{(a)(a) : 0, (a)(b) : 2, (a)(c) : 2, (b)(a) :$

$1, (b)(b) : 3, (b)(c) : 3, (c)(a) : 0, (c)(b) : 1, (c)(c) : 1, (a, b) : 3, (a, c) : 1, (b, c) : 3\}$, $L_2 = \{(b)(b), (b)(c), (a, b), (b, c)\}$ is obtained.

For $j = 3$ onwards, L_{j-1} is joined with itself to obtain C_j. Two sequences s and s' are joined if and only if the sequences obtained by deleting the first item in s and the last item in s' are the same. The new sequence t is the sequence s extended with the last item in s', and the last item in s' is added the same way to s as it was in s'. For example, $(b)(b)$ is joined with (b, c) to obtain $(b)(b, c)$, whereas $(b)(b)$ does not join with (a, c) as the remaining sequences after deleting the first and last items in $(b)(b)$ and (a, c) respectively, i.e. (b) and (a), are not the same. Further, the apriori pruning is applied to C_j and then the support computation is performed. For example, we generate C_3 and perform support computation, $C_3 = \{(b)(b, c) : 3, (b)(b)(c) : 1\}$, thus obtaining $L_3 = \{(b)(b, c)\}$.

The algorithm stops when no more frequent sequences can be discovered or no more candidate sequences can be generated.

Gupta and Han [14] consider the major weaknesses of the GSP algorithm to be: (1) generating a huge number of candidate sequences and (2) multiple scans of the input database.

2.3.1.2 SPADE

The SPADE algorithm [36] assumes that the input database is in the form of vertical id-lists. The vertical id-lists for the sample database of Figure 2.1 are shown in Table 2.2. The search space exploration in SPADE is either breadth-first or depth-first, and the support computation is done using temporal id-list joins. We first give a few definitions.

Definition 2.8 (**S-extension).** Given a sequence $s = \langle s_1, \ldots, s_q \rangle$ and an item $\{x\}$, a sequence t is called an S-extension of s if $\{x\}$ is added to s as a new element, that is $t = \langle s.\{x\} \rangle$.

For example, given a sequence $s = \langle (a)(b,c) \rangle$ and an item $\{d\}$, an s-extension of s is $\langle (a)(b,c)(d) \rangle$.

Definition 2.9 (**I-extension).** Given a sequence $s = \langle s_1, \ldots, s_q \rangle$ and an item $\{x\}$, a sequence t is called an I-extension of s if $\{x\}$ is added as an item to the last element s_q in s, that is $t = \langle s_1, \ldots, s_q \cup \{x\} \rangle$. Note that $\{x\}$ can only be added to s_q if it is lexicographically greater than all the items in s_q.

For example, given a sequence $s = \langle (a)(b,c) \rangle$ and an item $\{d\}$, an I-extension of s is $\langle (a)(b,c,d) \rangle$. However, s can not be I-extended with item $\{a\}$, as $\{a\}$ is not lexicographically greater than all the items in the last element of s. Next, we define the notion of prefix.

Definition 2.10 (**Prefix).** Given two sequences $s = \langle s_1, \ldots, s_q \rangle$ and $t = \langle t_1, \ldots, t_r \rangle$, s is called a prefix of t, if for $k = 1, \ldots, q-1$, $s_k = t_k$, and for $k = q$, $s_k \subseteq t_k$, and all the items in $(t_k - s_k)$ are lexicographically after those in s_k.

Note that if t is an S- or I-extension of s, then s is a prefix of t. For example, for a sequence $s = \langle (a)(a,b,d)(b,c) \rangle$, sequences such as $\langle (a) \rangle$, $\langle (a)(a,b) \rangle$ and $\langle (a)(a,b,d)(b) \rangle$ are all prefixes of s, whereas $\langle (a)(b,d) \rangle$ and $\langle (a,b,d) \rangle$ are not. For a sequence s, a prefix of length k is called a k-prefix of s.

We now give an overview of (the breadth-first variant of) SPADE. First, L_1 is computed. Then, for $j = 2$ onwards, L_{j-1} is both S- and I-extended with L_1 to generate C_j. Next, apriori pruning is applied to C_j and then support computation is performed using temporal id-list joins (which we explain later) and thus, L_j is obtained.

TABLE 2.2: Vertical id-list for the items in the database of Figure 2.1.

Vertical id-lists							
(a)		(b)		(c)		(d)	
σ	eid	σ	eid	σ	eid	σ	eid
1	1	1	1	1	3	1	1
2	4	1	3	1	6	1	3
3	5	1	6	2	4	1	6
		2	2	3	7	3	7
		2	4				
		3	5				
		3	7				

TABLE 2.3: Temporal id-list join using Vertical id-list in Table 2.2.

Temporal id-list joins							
$(a)(b)$		$(b)(a)$		(a,b)		$(a,b)(c)$	
σ	eid	σ	eid	σ	eid	σ	eid
1	3	2	4	1	1	1	3
3	7			2	4	3	7
				3	5		

The algorithm stops when no more frequent sequences can be discovered or no more candidate sequences can be generated, and outputs all frequent sequences.

As already mentioned, the support computation in SPADE is using temporal id-list joins. We briefly explain the working of a temporal id-list join for the S- and I-extension cases. For the S-extension case, suppose that s is S-extended with an item $\{x\}$, then a source σ supports the S-extension of s, if an eid in the id-list of $\{x\}$ can be found which is greater than the smallest eid in the id-list of s for source

σ. For example, when computing the support of a sequence $\langle(a)(b)\rangle$, we pick the smallest *eid* for source σ_1, namely $eid = 1$, and find the smallest *eid* in the id-list of (b) for source σ_1 which is greater than 1, namely $eid = 3$. If such an *eid* can be found, it means that source supports the sequnece. In case of I-extensions, suppose that s is I-extended with an item $\{x\}$, then the source σ and the *eid* both have to match in the id-list of s and $\{x\}$, which means that both the sequence s and the I-extension are present in the same event in that source. For example, for σ_1 and a sequence $\langle(a, b)\rangle$, both the items (a) and (b) are present in the event corresponding to $eid = 1$, and thus σ_1 supports the sequence $\langle(a, b)\rangle$. We show temporal joins for some sample sequences for the vertical id-lists of Table 2.2 in Table 2.3. For more details, see [36].

SPADE uses a lattice theoretic approach for search space exploration and thus decomposes the search space into sub-lattices which can be processed independently in memory. Further, SPADE only needs three database scans to find the complete set of sequential patterns. If some preprocessed information is available, the complete set of sequential patterns can be found in a single scan of the database. SPADE is shown to be scalable under different parameter settings.

2.3.1.3 SPAM

SPAM [34] is another important classical SPM algorithm that explores the search space in a depth-first manner. SPAM constructs a lexicographic sequence tree where every node contains a sequence s, and the children of each node are the S- or I-extensions of s. If at any point, an S- or I-extension of a sequence s is not frequent, the item is not considered for extending any of the extensions of s due to the apriori property. The support computation in SPAM uses a bitmap data structure (similar to vertical id-lists in SPADE) where every element in a sequence is presented as a

bit, and the bits in a bitmap are set in a way that the support computation is simply a bitwise AND operation on the bitmaps.

SPAM is similar to SPADE as it is also a depth-first algorithm and uses a bitmap data structure similar to the vertical id-lists in SPADE. The idea of bitwise AND joins in SPAM is also similar to that of temporal id-list joins in SPADE. However, in SPAM it is assumed that the bitmap data structure can fit into main memory. In the experimental studies, SPAM is shown to be upto 2.5 times faster than SPADE but is also shown to require upto 5 to 20 times more memory than SPADE. Thus, the choice between the two algorithms is rather a space-time trade-off [15].

This completes the overview of the popular candidate generation algorithms. We now focus on pattern growth algorithms.

2.3.2 Pattern Growth

Han et al. [39] first proposed the idea of pattern growth and introduced the FreeSpan algorithm. The idea was to avoid the expensive step of candidate generation by constructing projected databases and then mine those projected databases for finding frequent sequential patterns. However, the original approach to constructing projected databases also proved to be CPU and memory intensive. In a subsequent study, Pei et al. [40] improved the database projection step and introduced the PrefixSpan algorithm. We first give a few definitions and then give an overview of PrefixSpan.

Definition 2.11 (Weak-prefix). Given two sequences $s = \langle s_1, \ldots, s_q \rangle$ and $t = \langle t_1, \ldots, t_r \rangle$, 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.

The definition of weak-prefix is similar to that of prefix (recall Definition 2.10) as both match a sequence s in a sequence t, however, weak-prefix allows gaps whilst matching s in t whereas prefix doesn't.

Definition 2.12 (**Weak-suffix).** Given two sequences $s = \langle s_1, \ldots, s_q \rangle$ and $t = \langle t_1, \ldots, t_r \rangle$, where s is a weak-prefix of t, a sequence $u = \langle u_q, \ldots, u_r) \rangle$ is the weak-suffix of s, where $u_k = (t_k - s_k)$ for $k = q$, and $u_k = t_k$, for $k = (q+1), \ldots, r$.

Definition 2.13 (**Projected Database).** Given a database D in the form of source sequences and a sequence s, an s-projected database D^s is the collection of weak-suffixes in the source sequences in D with respect to the weak-prefix s.

For example, for a sequence $s = \langle (a)(a,b,d)(b,c)(d,e) \rangle$, sequences such as $\langle (a) \rangle$ and $\langle (a)(b,d) \rangle$ 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 in s are $\langle (a,b,d)(b,c)(d,e) \rangle$ and $\langle (-,d)(b,c)(d,e) \rangle$ respectively, where '$-$' means that one or more items in the event are part of the weak-prefix. In what follows, we sometimes use the terms prefix and suffix for weak-prefix and weak-suffix when it is clear from the context. An $\langle (a) \rangle$-projected database for the sample database of Figure 2.1 is $\{\langle (-,b,d)(b,c,d)(b,c,d) \rangle, \langle (-,b,c) \rangle, \langle (-,b)(b,c,d) \rangle\}$.

PrefixSpan first discovers the set of frequent 1-sequences L_1. Next, the idea is to partition the complete set of sequential patterns into as many subsets as the number of frequent 1-sequences. For example, if $L_1 = \{(a), (b), (c)\}$ for the sample database of Figure 2.1, then each of the frequent sequential patterns must begin with an item in L_1. Thus, considering each 1-sequence s as a prefix, the corresponding projected databases are built and mined recursively to mine the complete set of sequential patterns. For example, if $\{b\}$ is a frequent item in the $\langle (a) \rangle$-projected database of Figure 2.1, then an $\langle (a)(b) \rangle$-projected database can be constructed as

$\{\langle(-, c, d)(b, c, d)\rangle, \langle\rangle, \langle(-, c, d)\rangle\}$ and can be mined recursively to discover frequent sequences having a prefix $\langle(a)(b)\rangle$. This recursive mining process stops when no more projected databases can be constructed or no more sequential patterns can be found. See Pei et al. [40] for more details.

PrefixSpan is different from candidate generation algorithms as no candidate sequences are generated and frequent prefixes are grown in a systematic way to find the set of frequent sequences. Although the projected databases keep shrinking, the major cost in PrefixSpan is the construction of projected databases for which authors propose optimizations like pseudo-projection and bi-level projection which work only if the projected databases can fit into main memory. In experimental study of [40], PrefixSpan is shown to outperform GSP and FreeSpan. However, the relative performance of PrefixSpan compared to SPADE and SPAM depends on the settings of various parameters.

This concludes our discussion of some of the important classical SPM algorithms. We have discussed the main ideas in each algorithm and how these algorithms compare with each other under different parameter settings. Whilst some of the ideas proposed in individual algorithms are similar to each other, the algorithms can broadly be classified based on the candidate generation or the pattern growth framework. In the next section, we compare and contrast the main features of SPM algorithms based on these two classes (i.e. candidate generation and pattern growth).

2.4 Comparison of SPM Algorithms

We first focus on candidate generation algorithms.

2.4.1 Candidate Generation

As we discuss the important features of candidate generation algorithms, we focus on the candidate generation algorithms discussed in the previous section, i.e. GSP [38], SPADE [36] and SPAM [34].

(a) Candidate generation algorithms traverse the search space in a breadth-first or in a depth-first manner. The breadth-first algorithms, namely GSP and the breadth-first variant of SPADE, can make full use of the apriori pruning. The depth-first algorithms, namely SPAM and depth-first variant of SPADE, can only make partial use of apriori pruning. This happens as when processing a candidate j-sequence, all of its $(j-1)$-subsequences might not have been processed and thus, the information needed for apriori pruning may not be available.

(b) Candidate generation algorithms explicitly generate candidate sequences for search space exploration. Thus, GSP joins L_{j-1} with itself to obtain C_j, whereas SPADE and SPAM join L_{j-1} with L_1 to get C_j.

(c) The support computation details are also different. Typically, an algorithm also has an associated data structure for the support computation task. For example, GSP builds a hashtree using the candidate sequences and then for each source sequence explores the hashtree systematically to compute the support of candidate sequences. SPADE transforms the input database to a vertical id-lists format, and uses temporal id-list joins for the purpose of support computation. SPAM has a bitmap representation for each source sequence, and uses bitwise AND operations for the purpose of support computation.

(d) In practice, most candidate generation algorithms scan the input database multiple times for the purpose of support computation. The algorithms which

require fewer scans store the input database in main memory at some point during the execution of the algorithm. For example, to find frequent k-sequences, GSP is required to scan the input database k times. In contrast, SPADE only needs three database scans, i.e. one each for finding frequent 1- and 2-sequences and one scan for finding frequent 3-sequences onward; and uses vertical id-lists as in memory data structure for the purpose. SPAM does not scan the database multiple times as it assumes that the database can fit into main memory.

2.4.2 Pattern Growth

Pattern growth algorithms are based on the idea of database projection. We give some common characteristics of the pattern growth algorithms and focus on PrefixSpan in particular.

(a) In general, pattern growth algorithms explore the search space in a depth-first manner, and work on the idea of search space partitioning, e.g. PrefixSpan does this by creating projected databases. SPADE (a candidate generation algorithm) also does this by using the notion of equivalence classes.

(b) These algorithms do not explicitly generate candidate sequences, and rather use frequent sequences starting from frequent 1-sequences as prefixes to generate projected databases, and mine those projected databases to find possible frequent extensions of a frequent sequence recursively.

(c) Pattern growth algorithms work under the assumption that the database can fit into main memory and thus do not need to scan the database multiple times. Further, the algorithms usually do not physically create projected databases,

instead they use pointers to move into projected databases, e.g. using ideas like pseudo-projection or bi-level projection in PrefixSpan.

We have summarized the distinctive features for SPM algorithms based on the candidate generation and the pattern growth frameworks. In summary, whilst some of the properties of individual algorithms in each class overlap with others, SPM algorithms are distinguished on the basis of search space exploration and support computation details.

2.5 Alternative SPM Formulations

Classical SPM algorithms discover the complete set of frequent sequential patterns. There are two main issues with this approach. Firstly, the reported sequential patterns may be huge in number and only a few of them might be of interest. Secondly, finding the complete set of sequential patterns is CPU and memory intensive. Many alternative SPM formulations have been proposed in the literature that address the above mentioned issues. Gupta and Han [14] give a detailed account of the alternative SPM formulations that have been proposed in literature. We briefly review a few important formulations here.

2.5.1 Constrained SPM

Constrained SPM algorithms are SPM algorithms that find the set of sequences that meet some user-defined constraint. In classical SPM, the only user-defined constraint is the minimum support threshold. However, many other constraints have been proposed in the literature, for example: Srikant and Agrawal [38] define time constraints, a sliding time window and a user-defined taxonomy; Yun [41] proposes

weight constraint in order to reduce uninteresting sequential patterns; and Garofalakis et al. [42] propose regular expression constraints. Gupta and Han [14] observe that constraints could be *monotonic*, i.e. if a sequence satisfies a constraint so do any of its supersequences; or *anti-monotonic*, i.e. if a sequence does not satisfy a constraint neither do any of its supersequences; or of other types. Pei et al. [43] discuss several classes of constraints, for example:

- item constraint: that is, a subset of items that should or should not be present in a pattern.
- super-pattern constraint: that is, a pattern must contain at least one of a particular set of patterns as sub-pattern.
- time constraint: that is, the time-stamp difference between the first and the last element in a sequential pattern should be more or less than a user-specified time span.

See Pei et al. [43] for a detailed discussion on constrained SPM.

2.5.2 Closed SPM

We now discuss an important class of sequential patterns, namely closed sequential patterns. Recall from Definition 2.5 that a sequential pattern s is closed if there is no supersequence of s that has the same support as s. Yan et al. [35] argue that mining closed sequential patterns eliminates the redundant sequential patterns from the set of frequent sequential patterns, and they still have the same expressive power as the frequent sequential patterns. Consider for example a sample database having one source sequence of the form $\langle(s_1)\dots(s_{100})\rangle$ and assume a minimum support threshold $\theta = 1$. Then, the complete set of sequential patterns will contain 2^{100}

frequent sequences of which all will be redundant except the one which is closed. Thus, closed sequential patterns are much more compact than the frequent sequential pattern and arguably, still have the same expressive power.

Classical SPM formulations usually assume that the data is recorded deterministically. We now discuss a class of SPM formulations that assumes noise or uncertainty in data.

2.5.3 SPM from Noisy or Uncertain Data

Yang et al. [44] consider biological sequence data and argue that due to the presence of noise in the data, an observed item may not be the actual item. For example, mutations in an amino acid have a non-zero probability of occurrence with little or no impact on the biological function of the protein. If an amino acid N can mutate to D with little or no impact on the biological function of the protein, if an observed value is D in a protein sequence, the true value may be either of N or D. Yang et al. propose a compatibility matrix that captures the likelihood of an observed value of being the true value. For example, an observed value D may mean that it may be either of N or D with probabilities 0.9 and 0.1 respectively. Then, Yang et al. propose a match matrix to compute the probability that a pattern is supported by a sequence. Yang et al. give algorithms to compute the match of a sequential pattern and show the effectiveness of the match model with the help of experiments.

Sun et al. [45] consider a scenario in telecommunication network fault analysis where the time-stamps for a specific type of events (alarms) are uncertain. This is due to the reason that whilst the time-stamps for the rest of the events are recorded as soon as they occur, the readings for some kind of events are recorded only after a time interval of upto Δ. Suppose that at time t, alarm u is recorded after a time interval Δ. We know that u occurred some time in the interval $[t - \Delta, t]$ and thus,

the time-stamp for u is uncertain. Suppose that the time-stamp for another event v is recorded with certainty and is $t - \Delta/2$. Then, the order between u and v is uncertain. Sun et al. propose the notion of *precise support*, which is different from the classical support of a sequential pattern and give an algorithm similar to the apriori algorithm to find frequent sequential patterns from data with uncertain event ordering.

In a very recent work, Zhao et al. [46] propose mining probabilistically frequent sequential patterns from uncertain data. Note that this work is subsequent to the publication of our papers [47–50]. Zhao et al. consider two kinds of uncertainty models. Firstly, the sequence-level uncertainty model where an instance of a sequence exists with some probability. Secondly, the element-level uncertainty model where uncertainty is present in the events of a sequence. Zhao et al. give motivations to justify the two uncertainty models and define a probabilistic frequent sequential pattern for both the uncertainty models. Next, they extend PrefixSpan to work with the proposed uncertainty models and propose sequence-level U-PrefixSpan and element-level U-PrefixSpan. Zhao et al. demonstrate the scalability of the algorithms the propose, with the help of an experimental study. Finally, Zhao et al. demonstrate the usefulness of the element-level uncertainty model for RFID trajectory mining using real-life data.

2.6 SPM Applications

The SPM problem has been studied in a variety of application domains [18, 36, 51]. For example, sequential patterns have been used in recommendation systems for suggesting a product to a potential customer, for predicting future trends in stock market analysis, for fault analysis in detecting sequences of events that lead to a failure, etc. We briefly review some of the applications of SPM here.

Sequential pattern mining has been studied for many applications in bio-informatics, e.g. for protein function prediction and protein fold recognition, for analysing gene expression data and for motif discovery in DNA sequences. Biological sequences are different from market basket data as in biological sequences: (1) the alphabet size is small (2) sequences are long and (3) they have gaps, i.e. don't care, of arbitrary length. Wang et al. [52] argue that mining biological sequences results in an explosion in the number of sequential patterns and that the classical SPM algorithms are not designed to handle this kind of explosion. Wang et al. propose a two-phase algorithm to address the explosion problem. Typically, biological sequences are of the form $\langle s_1 \ * \ldots * \ s_q \rangle$ where s_i is a small region of consecutive items (segment) and $*$ represents a gap of arbitrary length. The algorithm works in two phases: the first is the segment phase which finds all frequent segments, followed by the pattern phase where segments are grown using frequent segments (not items) discovered in the segment phase. The use of segment growth over item growth results in an overall speed up for the algorithm. Wang et al. propose pruning techniques similar to the apriori pruning to gain speed up, and demonstrate the superiority of their approach to PrefixSpan with the help of experiments.

Wang et al. [53] propose protein function prediction using SPM. Wang et al. adopt a two phase approach for protein function prediction. In the first phase, they mine a known protein sequence dataset and find frequent segments, similar to [52]. In the second phase, they build a classifier to predict the function of protein sequences. Thus, whilst Wang et al. use segment growth similar to [52] when generating frequent sequences, one important distinction is that they consider mining closed sequential patterns only in order to improve the accuracy of the classifier. Wang et al. demonstrate the effectiveness of the protein function prediction algorithm with the help of experiments.

In another work, Exarchos et al. [18] employ SPM for protein fold recognition. The

idea is to assign a protein sequence whose structure is unknown, to one of the folds (recall that a fold is 3D structure of a protein) in order to determine the protein function. Exarchos et al. use the extracted sequential patterns from all folds for the purpose and check if each of the frequent patterns is contained in the protein. A score is maintained against each fold, which means that the protein belongs to this particular fold. The protein is assigned to the fold that has the highest score for that protein.

Web usage mining is another important application of SPM. Mobasher et al. [16] suggest that web usage data can be used for web personalization. For example, common navigation paths might suggest the next web page that will be accessed by the user and predictive tasks such as web pre-fetching can be performed in order to improve the navigation experience of users. Mobasher et al. propose a recommendation algorithm that works by finding frequent sequential patterns in the user navigation data. Typically, the algorithm computes the frequent patterns and the confidence in the patterns. Then, for frequent sequential patterns of length k, if a user has followed the $(k-1)$-prefix of a sequential pattern, the next web page that will be accessed by the user can be predicted with some confidence.

Zaki [36] uses sequential patterns to predict plan failures. TRIP is a system that is used to develop evacuation plans from a small island. During plan development, TRIP simulates plan success or failure in real life situations like unfavourable weather conditions or in case of vehicle breakdown. When a plan fails, the sequence of steps that led to plan failure can be analysed in order to improve the plan. Zaki uses SPADE to mine such plans. Zaki observes that many repetitive patterns are reported in plan mining, and proposes pruning techniques to find only interesting patterns.

Ishio et al. [54] proposed mining sequential patterns to detect *cross-cutting concerns*

in Java programs. A concern is a particular behaviour needed for a computer program such as database interaction; and a cross-cutting concern is a concern that effects other concerns, for example two pieces of code in the same module of an application interacting with the database. A coding pattern can be seen as a concern, i.e. a sequence of method calls and control statements to implement a specific behaviour. Ishio et al. define a set of rules to convert a Java program to a sequence database and use PrefixSpan to extract frequent concerns. They claim that they are able to detect cross-cutting concerns successfully unless the order of method calls has been modified. One of the limitations of this work pointed out by Ishio et al. is that if there is a long segment of copy-and-paste code, the algorithm will not be able to detect it as it may not be frequent. One possible solution to this problem could be to assign weights in accordance with the pattern length [41].

Sequential patterns have also been used for text (document) categorization. Jaillet et al. [17] define the text categorization task as follows. A boolean value 'true' is assigned to a (document, category) pair if the document belongs to that category, and 'false' otherwise. The transformation from a text document to a sequence database is as follows. Each document corresponds to a source, and each sentence in a document is considered as an event, and the time-stamps are assigned to sentences in the order as they appear in the document. Jaillet et al. propose a two-phase approach to the problem. First, they find frequent sequential patterns in the document and extract the sequential patterns along with the associated confidence values. Then, they classify the documents using the discovered sequential patterns. A document is categorized based on the k frequent sequential patterns having highest confidence for the document, and then the best match for the k sequential patterns in document categories is the category of the document.

This concludes our discussion of SPM applications. We have briefly reviewed some of the applications of SPM and it can be seen that SPM can be applied to a variety

of application domains. For a more comprehensive overview of SPM applications, see Gupta and Han [13].

2.7 Summary

We have given an overview of association rule mining, and have focussed on sequential pattern mining in particular. We have formally defined the SPM problem and have discussed the computational complexity of computing all frequent sequences. We have categorized SPM algorithms into two classes, based on the candidate generation and the pattern growth frameworks. We have considered three algorithms based on the candidate generation framework, namely GSP, SPADE and SPAM, and one algorithm based on the pattern growth framework, namely PrefixSpan. We have given an overview of each of these algorithms and have contrasted their main characteristics. We have considered some alternative SPM formulations as well, and have reviewed work on SPM using noisy or uncertain data. Finally, we have discussed some applications of SPM.

Chapter 3

Probabilistic Databases

Probabilistic databases were first proposed by Cavallo and Pittarelli [55], and have been revitalized recently by Dalvi and Suciu [56]. Suciu and Dalvi [24] highlight two important factors in the recent interest in probabilistic databases. First, a wide range of newly emerging applications need to handle imprecisions in data, e.g. managing sensor and RFID data, information extraction, deduplication, and privacy preservation. Second, some recent advancements in handling probabilistic data along with some old results could help in devising practical probabilistic database management systems.

According to Boulos et al. [57], a probabilistic database is a finite set of *possible worlds* where every possible world has some probability of occurrence. Each possible world is a deterministic database but only one world from the set of possible worlds is the 'true' world and it is not known which one. The probability of a possible world is the degree of belief in a possible world being the true world. There have been many studies on probabilistic database systems [22]. Dalvi et al. [1] specify *probabilistic inference*, namely to compute the posterior probability distribution of a query variable given some evidence, as one of the key challenges for a probabilistic

database system. Recently, researchers have focused on special cases of probabilistic inference that occur during query evaluation, e.g. lineage based systems [21] and top-k query evaluation [3].

In this chapter, we first give motivations for probabilistic databases with the help of two case studies. Then, we review representative probabilistic data models that have widely been used in the literature. We briefly discuss query evaluation for a probabilistic database system, and also give a brief overview of representative probabilistic database systems that have been proposed in the literature. Modelling uncertain data as probabilistic databases has led to several ranking and data mining problems being studied in the context of uncertain data. We focus on two such problems; first, we briefly review the top-k problem (a ranking problem) and then give a detailed account of the frequent itemset mining problem (a data mining problem) in the context of uncertain data.

3.1 Motivations for Probabilistic Databases

Typically, a probabilistic database system has to specify the following key details: (a) probabilistic data model (b) query evaluation using the chosen data model and (c) presentation of query results to the user. In the remainder of this section, we will refer to these "key details" as points (a), (b) and (c). We illustrate these concepts with the help of two case studies: (1) information extraction and (2) deduplication.

3.1.1 Information Extraction

The DbLife system [1] at Yahoo! extracts and manages structured information from the web about researchers in the database community. The information that the DbLife system [1] extracts from the web includes a researchers' name, affiliation,

Researcher			
rid	*name*	*affiliation*	Pr
r_1	Fred	U. Washington	0.3
r_2		U. Wisconsin	0.2
r_3		Y! Research	0.5
r_4	Sue	U. Washington	1.0
r_5	John	U. Wisconsin	0.7
r_6		U. Washington	0.3
r_7	Frank	Y! Research	0.9
r_8		M. Research	0.1

Service				
sid	*name*	*conference*	*role*	Pr
s_1	Fred	VLDB	Session Chair	0.2
s_2	Fred	VLDB	PC Member	0.8
s_3	John	SIGMOD	PC Member	0.7
s_4	John	VLDB	PC Member	0.7
s_5	Sue	SIGMOD	Chair	0.5

FIGURE 3.1: An example of a probabilistic database (reproduced from Dalvi et al. [1]).

publications and role (professional activities). In Figure 3.1, we show researcher's affiliations in the `Researcher` table and their roles in the `Service` table. Observe that a researcher's affiliation in `Researcher` is not unique (certain) although most will have only a single association. The uncertainty in the affiliation attribute is due to data being collected from multiple sources, of which some might be outdated or erroneous (this is the source of uncertainty). If information extractor relies only on the most recent data, there is a risk of missing valuable information. Thus, the information extractor introduces uncertainty in data by computing the confidence values (interpreted as probabilities) using an entity matching algorithm [1], which

Researcher			
rid	*name*	*affiliation*	Pr
r_1	Fred	U. Washington	0.3
r_2	Fred	U. Wisconsin	0.2
r_3	Fred	Y! Research	0.5
r_4	Sue	U. Washington	1.0
r_5	John	U. Wisconsin	0.7
r_6	John	U. Washington	0.3
r_7	Frank	Y! Research	0.9
r_8	Frank	M. Research	0.1

Rule	
γ	*ExclusionRule*
γ_1	$\{r_1, r_2, r_3\}$
γ_2	$\{r_4\}$
γ_3	$\{r_5, r_6\}$
γ_4	$\{r_7, r_8\}$

FIGURE 3.2: Attribute-level uncertain `Researcher` table from Figure 3.1 transformed to x-tuples.

gives answers to questions such as "is Fred on webpage A is the same as Fred on webpage B?"

We now focus on uncertain data modelling. One may view the affiliation attribute in the `Researcher` table as being the uncertain attribute (as we will see in Section 3.2.1, this can be modelled using attribute-level uncertainty). We now introduce some dependencies in the data as we define two constraints: (1) a name may have multiple values for the affiliation attribute, of which only one can be true, thus values in affiliation attribute are mutually exclusive; and (2) the values for the affiliation attribute for different name values are independent of each other.

We discuss the notion of dependence in uncertain data modelling in Section 3.2.3 using the x-relations model. An alternative representation of the `Researcher` table using the x-relations model (Section 3.2.3) is presented in Figure 3.2. In Figure 3.2, each tuple in the `Researcher` table is a separate uncertain tuple, and constraints are defined in the `Rule` table using exclusion rules where the tuples in a rule are mutually exclusive and the tuples in different rules appear independent of each

other. The records in the `Service` table (Figure 3.1) are also extracted from the web using the conference webpages and are not precise either. Thus, the records in the `Service` table have existential probabilities (as we will see in Section 3.2.2, this kind of uncertainty can be modelled using tuple-level uncertainty). We also assume that the records in the `Service` table are independent of each other. This completes our discussion on a probabilistic data model (which is point (a)) for an information extraction system.

Once the probabilistic data model is determined, probabilistic database system can evaluate queries (point (b)) like "find all affiliations with more than 12 SIGMOD and VLDB PC Members" [1] and finally, the results can be presented to the user in a meaningful way (point (c)), e.g. using top-k or some other scheme.

3.1.2 Deduplication

Another important application that handles imprecise data is that of *deduplication*, i.e. identifying multiple records in a database that may represent the same real-world entity. Hassanzadeh and Miller [2] argue that not only is it expensive to perform data cleaning (i.e. to accurately merge duplicates in data) manually, but also that complete deduplication may result in valuable information loss. The solution to this problem is to keep all the data and to introduce a notion of uncertainty for tuples that are potential duplicates. Next, the potential duplicates are grouped together in clusters and uncertainties (probabilities) are assigned to various alternatives based on some similarity measure.

A sample "dirty" database is shown in Figure 3.3. The tuples with the same *cid* value in the `Company` table in Figure 3.3 are potential duplicates, and have been assigned probabilities accordingly. As the tuples in a cluster are potential duplicates, they are assumed to be mutually exclusive. In addition, the tuples in different clusters

`Company`

tid	*name*	*emp#*	*hq*	*cid*	Pr
t_1	Altera Corporation	NY	6K	c_1	0.267
t_2	Altersa Corporation	New York	5K	c_1	0.247
t_3	lAtera Croporation	New York	5K	c_1	0.224
t_4	Altera Corporation	NY, NY	6K	c_1	0.262
t_5	ALTEL Corporatio	Albany, NY	2K	c_2	0.214
t_6	ALLTEL Corporation	Albany	3K	c_2	0.208
t_7	ALLTLE Corporation	Albany	3K	c_2	0.192
t_8	Alterel Coporation	NY	5K	c_2	0.184
t_9	Alterel Corporation	Albany, NY	2K	c_2	0.202

`Product`

pid	*product*	*tidFK*	*cidFK*	*cid*	Pr
p_1	MaxLink 300	t_1	c_1	c_3	0.350
p_2	MaxLink 300	t_8	c_2	c_3	0.350
p_3	MaxLnk 300	t_4	c_1	c_3	0.300
p_4	Smart Connect	t_6	c_2	c_4	1.0

`Price`

rid	*product*	*price*	*cid*	Pr
r_1	MaxLink 300	285	c_5	0.8
r_2	MaxLink 300	100	c_5	0.2

FIGURE 3.3: An sample dirty database with `Company`, `Product` and `Price` relations (reproduced from Hassanzadeh and Miller [2]).

are assumed to be independent of each other; the intuition for this is that the errors in different real world entities are introduced independently of each other. This corresponds to the *block independent-disjoint* model by Dalvi et al. [1], which says that the tuples in a block (cluster) are disjoint and the tuples in different blocks are independent (point (a)); the block independent-disjoint model is similar to the x-relations model and we briefly discuss this in Section 3.2.3.

Once the probabilistic data model is defined, different queries can be run on the probabilistic data (point (b)) and the query results be presented to the user in a meaningful way (point (c)), as we have already discussed in Section 3.1.1.

Thus, we have motivated the case for probabilistic databases with the help of two case studies. We have seen that uncertainties in data are introduced due to various reasons and that some independence assumptions are made in order to model uncertain data.

We now discuss probabilistic data models.

3.2 Probabilistic Data Models

Sarkar and Dey [58] suggest that the kind of uncertainty that arises due to ambiguity—when choices among several precise alternatives cannot be perfectly resolved—is better modelled using probabilities, and discuss families of uncertainty models that have been proposed in literature. A probabilistic data model describes a probability distribution over the set of possible worlds. A model which is *complete* enumerates the full set of possible worlds and is able to capture any kind of correlations in data; but there can be an exponential number of possible worlds. In practice, some simplifying assumptions are made, e.g. assuming independence among tuples. Thus, the simplest model is the tuple-level uncertainty model [56].

TABLE 3.1: A sample attribute-level uncertain database (reproduced from Cormode et al. [3]).

rid	W
r_1	$(a : 0.4)(b : 0.6)$
r_2	$(c : 0.6)(d : 0.4)$
r_3	$(e : 1.0)$

We now review some uncertainty models in literature.

3.2.1 Attribute-level Uncertainty Model

A probabilistic database D^p is a set of records $(r_1, \ldots, r_n)$, where each record has one attribute whose value is uncertain along with other attributes that are certain. The uncertain attribute in each record is described by a probability distribution W over a set of values. The distributions W are assumed to be independent of each other. A possible world D^* of D^p is generated by taking each record in turn and by selecting one value from the distribution of values for the uncertain attribute. The probability of a possible world D^* is computed as $\Pr[D^*] = \prod_{i=1}^{n} \Pr_{W_i}[x_i]$, where x_i denotes the value for the uncertain attribute in r_i. The complete set of possible worlds is obtained by enumerating all such possible combinations. A sample attribute-level uncertain database is shown in Table 3.1, and the set of possible worlds for the database of Table 3.1 is in Table 3.2.

3.2.2 Tuple-level Uncertainty Model

A probabilistic database D^p is a set of records $(r_1, \ldots, r_n)$, where each record has an existential probability. For every record r_i, there can be two kinds of possible

TABLE 3.2: The set of possible worlds for the database of Table 3.1 along with their probabilities.

D^*	r_1	r_2	r_3	$\Pr(D^*)$
D_1^*	(a)	(c)	(e)	$0.4 \times 0.6 \times 1 = 0.24$
D_2^*	(a)	(d)	(e)	$0.4 \times 0.4 \times 1 = 0.16$
D_3^*	(b)	(c)	(e)	$0.6 \times 0.6 \times 1 = 0.36$
D_4^*	(b)	(d)	(e)	$0.6 \times 0.4 \times 1 = 0.24$

(Reproduced from Cormode et al. [3])

TABLE 3.3: A sample tuple-level uncertain database.

rid	r	Pr
r_1	a	0.6
r_2	b	0.3

worlds: one where r_i appears and the other where it does not. A possible world D^* of D^p is generated by considering every record in turn, and choosing whether the record does or does not appear in the possible world. It is assumed that the existence of the records in D^* is independent of each other. The probability of a possible world D^* is computed as follows:

$$\Pr[D^*] = \prod_{r_i \in D^*} \Pr(r_i) \prod_{r_j \notin D^*} (1 - \Pr(r_j)),$$

where $\Pr(r_i)$ is the probability that the record r_i appears in the possible world. The complete set of possible worlds is obtained by enumerating all such possible combinations. A sample tuple-level uncertain database is shown in Table 3.3, and the set of possible worlds for the database of Table 3.3 is in Table 3.4.

TABLE 3.4: The set of possible worlds for the database of Table 3.3 along with their probabilities.

D^*	$\Pr(D^*)$
$D_1^* = \langle\rangle$	$(1 - \Pr(r_1)) \times (1 - \Pr(r_2)) = 0.28$
$D_2^* = \{a\}$	$\Pr(r_1) \times (1 - \Pr(r_2)) = 0.42$
$D_3^* = \{b\}$	$(1 - \Pr(r_1)) \times \Pr(r_2) = 0.12$
$D_4^* = \{ab\}$	$\Pr(r_1) \times \Pr(r_2) = 0.18$

3.2.3 x-relations Model

The attribute-level or the tuple-level uncertainty models are considered basic models due to the assumption of independence. Recall that it is assumed in the attribute-level uncertainty model that the distributions for the uncertain attribute are stochastically independent of each other, and that the existence of tuples is independent of each other in the tuple-level uncertainty model. Thus, both the attribute-level and the tuple-level uncertainty models can not capture correlations in data which might be needed for some applications.

Cormode et al. [3] consider the *x-relations* model which is the tuple-level uncertainty model with dependencies in the form of exclusion rules. The idea is to capture some basic dependencies in data, although some simplifying assumptions have to be made, as arbitrary exclusion rules have been shown to have exponential computational complexity [3]. We now define the x-relations model. A probabilistic database D^p is a set of records $(r_1, \ldots, r_n)$ where each record has an existential probability (tuple-level uncertainty), and the dependencies are defined in the form of a set of rules $(\gamma_1, \ldots, \gamma_m)$ where each rule contains a set of records. It is assumed that the records in a rule are mutually exclusive but the records in different rules appear independently of each other. A rule γ may contain only one record, so we can assume that all the records appear in exactly one of the rules. The sum of the probabilities of all the records present in a rule must not exceed 1.

rid	*r*	Pr
r_1	a	0.4
r_2	b	0.5
r_3	c	1.0
r_4	d	0.5

γ	*Rule*
γ_1	$\{r_1\}$
γ_2	$\{r_2, r_4\}$
γ_3	$\{r_3\}$

FIGURE 3.4: An example of x-relations model (reproduced from Cormode et al. [3]).

TABLE 3.5: The set of possible worlds for the database of Figure 3.4 along with their probabilities.

D^*	$\Pr(D^*)$
$D_1^* = \{r_1, r_2, r_3\}$	$\Pr(r_1) \times \Pr(r_2) \times \Pr(r_3) = 0.2$
$D_2^* = \{r_1, r_3, r_4\}$	$\Pr(r_1) \times \Pr(r_3) \times \Pr(r_4) = 0.2$
$D_3^* = \{r_2, r_3\}$	$(1 - \Pr(r_1)) \times \Pr(r_2) \times \Pr(r_3) = 0.3$
$D_4^* = \{r_3, r_4\}$	$(1 - \Pr(r_1)) \times \Pr(r_3) \times \Pr(r_4) = 0.3$

(Reproduced from Cormode et al. [3])

An alternate name for the x-relations model is the *block independent-disjoint model* which assumes that the tuples in a block are disjoint and the tuples in different blocks are independent [1]. A possible world D^* of D^p is generated similar to the tuple-level uncertainty model with the restriction that only one of the tuples in an exclusion rule appears in a possible world, and the probability of a possible world is also computed similarly to the tuple-level uncertainty model. A sample uncertain database having x-tuples is shown in Figure 3.4, and the set of possible worlds for the database of Figure 3.4 is shown in Table 3.5.

This concludes our discussion of probabilistic data models. In summary, the choice of a probabilistic data model depends on the nature of uncertainty in data and the kind of assumptions that are to be made for the underlying uncertainty. We have discussed some uncertainty models and also their expressive power.

Once a probabilistic data model has been determined, a variety of queries could be run on the probabilistic data, as we briefly discuss in the following section.

3.3 Query Evaluation

Dalvi et al. [1] describe query evaluation as the most challenging aspect of a probabilistic database system. Dalvi and Suciu [59] suggest integrating probabilistic inference with the query computation plan. A query specifies a computational task in a declarative language that is then transformed into relational algebra using operations like select, join and duplicate elimination, and is called a *relational* plan. The correctness of a relational plan is established by the query optimizer by a static analysis of the plan, and a relational plan is *safe* if it computes the query results correctly. Dalvi and Suciu [59] show that under some assumptions, it is possible to show a complete dichotomy, i.e. each query is either in polynomial time (i.e. queries for which a safe plan exists) or #P-complete (i.e. queries for which a safe plan does not exists). Thus, the challenge for a query optimizer is to identify and use a safe plan if one exists, and use a probabilistic inference algorithm that approximates a #P-complete problem, otherwise.

After a query has been evaluated, query results are presented to the user in a meaningful way. In case of certain data, it is rather obvious that the results presented to the user should be ordered based on some attribute, for example score attribute in top-k. The choice is not so straightforward in case of uncertain data, as scores for example have associated probabilities in the uncertain case, and it is not clear whether the results should be ordered by score or by probabilities.

3.4 Top-k

Top-k and other ranking queries have been extensively studied in literature [19, 60–62]. Cormode et al. [3] describe the semantics of ranking queries on uncertain data and suggest that a ranking query should ideally have the following properties:

(a) *Exact-k:* The top-k list should contain exactly k items.

(b) *Containment:* The top-k should be a subset of top-$(k+1)$.

(c) *Unique-ranking:* Each item in top-k should have a unique rank, i.e. an item should not appear multiple times in top-k.

(d) *Stability:* If an item is made more probable or more important (by increasing score), it should not be removed from the top-k.

(e) *Value Invariance:* If we change the scores without effecting the relative ordering of items, the top-k should remain unchanged.

The above mentioned properties are all satisfied in case of certain data and intuitively this is how the top-k should behave in the uncertain case as well. Cormode et al. [3] show, with the help of examples, that the ranking definitions proposed in the literature for uncertain data do not necessarily satisfy all these properties.

Suppose that the items in the sample database of Figure 3.4 are mapped to scores as follows: $(a) = 100$, $(b) = 92$, $(c) = 80$ and $(d) = 70$. As discussed in Cormode et al. [3], the top-k problem can be formalized in many different ways:

- *U-topk:* that is [60], obtain top-k from each possible world and then, compute the probability of each distinct top-k set and report the most likely top-k. This definition is shown to violate the containment property, for example, for

Figure 3.4, the top-1 result is r_1 (r_1 has the highest probability of having the highest score in a random possible world) but the top-2 result is either (r_2, r_3) or (r_3, r_4). Thus, the top-2 is completely disjoint from top-1.

- *U-kRanks:* that is [60], obtain the tuple r_i at rank k in each possible world and then report the tuple most likely to be at rank k over all possible worlds. Although U-kRanks does not have the limitations of U-top$k$, it fails on unique ranking as a tuple may dominate multiple ranks at the same time. For example, for Figure 3.4, the top-3 result is (r_1, r_3, r_1) and thus, r_1 appears twice and r_2 does not.

- *PT-k:* that is [61], if a tuple r_i is at rank k, or better, in a possible world; then the top-k probability of a tuple is the probability that it is in the top-k over all possible worlds. The PT-k reports the set of all tuples whose top-k probability is at least a user-specified probability p. However, for a user-specified p, the top-k may not always contain k tuples thus, violating exact-k. For example, for $p = 0.4$ in Figure 3.4, the top-1 result is (r_1) and the top-2 and top-3 results are both (r_1, r_2, r_3).

- *Global-Topk:* that is [62], rank tuples by their top-k probability (as in PT-k), and then select top-k tuples. Thus, whilst Global-Top$k$ makes sure that exactly k tuples are reported, it fails on containment. For example, for Figure 3.4, the top-1 result is (r_1) and the top-2 result is (r_3, r_2).

Thus, with the help of above examples, Cormode et al. [3] argue that probabilistic inference is not straightforward as the choice between the score and the associated probabilities is complicated, and it appears as if no single definition of top-k is plausible for all applications. However, whilst the above-mentioned properties do not fully describe a ranking query, only a subset may be desirable for some specific applications in the uncertain case.

3.5 Probabilistic Database Systems

Many implementations of probabilistic database systems have been described in the literature. We briefly review two such representative systems: (1) MystiQ and (2) Trio. For a good overview on managing probabilistic data, see Aggarwal [19].

3.5.1 MystiQ

MystiQ [57] is a probabilistic database system that uses probabilistic query semantics to answer queries from data that is obtained from different data sources and is uncertain. As already mentioned, the uncertainty in data from multiple sources arises due to various reasons, for example the same data item may be presented differently in different data sources, or although different data sources may contain contradictory information, the user may still want to ask complex structural queries. If standard query semantics are used, most non-trivial queries will return an empty answer set. MystiQ uses probabilistic query semantics and returns a set of probabilistic answers to queries ranked by the probability of the presented answer being the right answer. The probabilities with the answers could be static (depending on factors like the underlying data sources) or dynamic depending on how well the tuples in the data match the approximate predicates in the query. In summary, MystiQ supports complex queries with approximate match predicate over inconsistent data and presents results as soft views, i.e. views with probabilistic tuples. For more details see [57].

3.5.2 Trio

The Trio system [21] is based on three abstractions: *data*, *data uncertainty*, and *lineage*, and thus the databases in Trio are called "Uncertainty-Lineage Databases".

The lineage of a tuple is defined as the information about its derivation. For example, a table `Drives` may be populated from online car registration databases; a tuple in `Drives` is then associated to its original source in Trio. As the confidence in a tuple comes directly from the confidence in its data source, lineage can help in understanding and explaining uncertainty. In Trio, the tuples in a relation may have one or more uncertain attributes with optional confidence values. For example, a sample tuple in (for example) a `Saw` table, which indicates what car a subject may have seen, could be of the form $(\text{Amy}, \{(\text{Honda} : 0.5)(\text{Toyota} : 0.3)(\text{Mazda} : 0.2)\})$, which means that Amy saw one of these cars with the given confidence values. Trio uses an extension of SQL and contains additional features for uncertainty and lineage, e.g. it supports queries like "find all witnesses contributing to Jimmy being a high suspect". See [21] for a more detailed account.

3.6 Frequent Itemset Mining using Probabilistic Data

Many data mining problems have recently been studied in the context of uncertain data [19]. We focus on frequent itemset mining in uncertain data, which was first proposed by Chui et al. [31], and since then has gained a great deal of attention from researchers [25–29]. The problem has been studied with respect to the uncertainty models described in Section 3.2. Researchers have considered two definitions of a frequent itemset, namely *expected support* and *probabilistic frequentness*. We consider each of these two in turn beginning with the former.

TABLE 3.6: A sample uncertain database D^p.

rid	W
r_1	$(a : 0.7)(b : 0.4)$
r_2	$(c : 0.6)(d : 0.3)$

3.6.1 Expected Frequent Itemset Mining

Chui et al. [31] motivate frequent itemset mining in probabilistic databases by arguing that in contrast with the certain case, there are situations when the presence of an item in a transaction is better recorded using a likelihood measure (i.e. by using probabilities). For example, in a medical dataset that contains recordings about symptoms in patients, the symptoms being subjective values, could more precisely be recorded using probabilities. Thus, a sample record about a patient in an uncertain dataset may look like $(1, \{(\text{depression} : 0.4),\ (\text{eatingdisorder} : 0.7)\})$ where 1 is the patient id, and the patient has two alternative diagnosis, namely "depression" and "eatingdisorder" with probabilities 0.4 and 0.7 respectively. It is assumed that the presence or absence of items in a transaction is independent of each other.

Following Chui et al. [31], we first define their frequent itemset mining in probabilistic databases problem. Let $\mathcal{I} = \{i_1, i_2, \ldots, i_q\}$ be the set of items. A set $U \subseteq \mathcal{I}$ is called an itemset. A probabilistic database $D^p = \{r_1, r_2, \ldots, r_n\}$ is a set of records where each record $r_i \in D$ is of the form (rid, W), where rid is a unique record identifier and W is a probability distribution over $\mathcal{I}$. The distribution W contains pairs of the form (i, c) where $i \in \mathcal{I}$ is an item, and $0 < c \leq 1$ is the associated confidence value. It is assumed that the presence of items in W is independent of each other. An example is shown in Table 3.6. The possible world semantics of D^p is as follows. For each item i having probability c_i of occurrence in a record r_k, there are two kinds of worlds: one where item i occurs and other, where it does not. A possible world

D^* of D^p is generated by considering each item in this way, i.e. the item may or may not appear in the possible world (this corresponds to the tuple-level uncertainty model, Section 3.2.2). The complete set of possible worlds, denoted by $PW(D^p)$, is obtained by taking all such possible combinations of items in which each item may or may not appear in the possible world. As the items are assumed to appear independently of each other in a possible world, the probability of a possible world D^* is computed as follows:

$$\Pr[D^*] = \prod_{k=1}^{n} \left(\prod_{i \in r_k} \Pr(i, r_k) * \prod_{j \notin r_k} (1 - \Pr(j, r_k)) \right), \tag{3.1}$$

where $\Pr(i, r_k)$ is the probability that the item i appears in the k-th record. For example, a possible world D^* for the sample database of Table 3.6 can be generated such that the items a and c appear in the possible world and the items b and d do not. The probability of D^* is, $\Pr[D^*] = 0.7 \times (1 - 0.4) \times 0.6 \times (1 - 0.3) = 0.1764$.

As every possible world is a deterministic database, the support of an itemset in a possible world is computed using Equation 2.1. For an itemset U, the *expected support* of U in D^p, denoted by $ES(U, D^p)$, is defined as follows:

$$ES(U, D^p) = \sum_{D^* \in PW(D^p)} \Pr[D^*] * Sup(U, D^*) \tag{3.2}$$

Alternatively, due to the assumption of independence between the items and the principle of linearity of expectation, we can compute the expected support of an itemset U in a probabilistic database D^p using the following equation:

$$ES(U, D^p) = \sum_{k=1}^{n} \prod_{i \in U} \Pr(i, r_k). \tag{3.3}$$

An itemset U is an *expected frequent itemset* if $ES(U, D^p)$ is greater or equal to some user-defined support threshold θ, $0 \leq \theta \leq n$. We now define the expected frequent

itemset mining problem.

Definition 3.1 (Expected Frequent Itemset Mining Problem). Given a probabilistic database D^p having n records, and a user-defined support threshold θ, $0 \leq \theta \leq n$, find all expected frequent itemsets in D^p.

Chui et al. [31] extend the classical apriori based frequent itemset mining algorithm to work with uncertain data, i.e. they embed the expected support computation sub-routine in the classical apriori algorithm and call it U-apriori. As the expected support computation is a relatively expensive task (in contrast with the support computation in classic frequent itemset mining), Chui et al. propose a framework for eliminating potential infrequent candidate itemsets without expected support computation, which uses a set of pruning techniques collectively referred to as *LGS-Trimming*. The study concludes with an experimental study that shows the scalability of the U-apriori algorithm in terms of the CPU cost, and the effectiveness of LGS-Trimming.

Considering that [31] is the very first work on the problem, the main contributions are the problem definition, the U-apriori algorithm and the proposed pruning technique LGS-Trimming. However, the synthetic datasets used to show the effectiveness of LGS-Trimming contain 75% of the items per record with very low existential probabilities, i.e. using a normal distribution with mean equal to 10% with 6% standard deviation. It is rather unlikely that LGS-Trimming will be equally effective when a high percentage of items in the dataset do not have very low existential probabilities associated with them, and we discuss this point later in this section.

In a subsequent work, Chui and Kao [63] improve on the idea of pruning from their earlier work [31] and propose a *decremental pruning* technique to eliminate potential infrequent candidate itemsets. The basic idea is to compute an upper bound on the expected support of candidate itemsets rather than performing the exact expected

support computation. Chui and Kao propose an optimization of the pruning rule which is to compute the upper bound only upto a point where it can be said with certainty that the itemset is frequent or otherwise. Another proposed optimization is to compute the upper bounds for itemsets having a common prefix together. The experimental study in [63] suggests that the decremental pruning is effective in eliminating potential infrequent candidate itemsets even when upto 75% of the items in each record have high probabilities, i.e. using a normal distribution with mean equal to 90% with 5% standard deviation. However, Aggarwal et al. [27] show, with the help of experiments, that the pruning techniques proposed by Chui et al. [31] and Chui and Kao [63] are very much dependent on the nature of the data and do not work well with datasets where items have high existential probabilities.

The initial studies on the expected frequent itemset mining problem [31, 63] are based on the apriori algorithm which is considered to be less effective than pattern growth (FP-Tree) [64] in classical frequent itemset mining. Thus, one natural research direction is to consider adapting the FP-Tree to the probabilistic case. Aggarwal et al. [27] highlight at least one fundamental problem with extending the FP-Tree to the probabilistic case and discuss various alternatives. The FP-Tree is a compressed data structure that represents the input database. Starting from the root node, each node contains an item along with the item count. The item count in the deterministic settings could be interpreted as the sum of the probabilities at a node in the probabilistic FP-Tree. The issue with storing the sum of the probabilities is that we lose the individual probability values which are needed for expected support computation. Another idea is to store the individual probabilities with every node and thus, to compute the exact expected support of an itemset, but this option is memory intensive. Thus, it is obvious that if we do not store all the individual probabilities at each node, *exact* expected support computation might not be a possibility.

We see two approaches in the literature to solve this problem. First, Leung et al. [26] propose the UF-Tree that is similar to the FP-Tree except that a new transaction is merged with an existing one only if the item and the item probability match in the new transaction. It is rather obvious that individual probabilities are not very likely to match with each other as the values are in the domain of real numbers in the range $(0, 1]$. Leung et al. propose a simple but intuitive solution to the probability matching problem which is to round off the probability values to a few decimal places, e.g. 2 decimals. Thus, a potential infinite number of values in the range $(0, 1]$ are reduced to a maximum of 10^2 possible values. Clearly, by rounding off the probability values, Leung et al. [26] are only able to compute an upper bound on the expected support of an itemset but the UF-Tree still has the FP-Tree compression effect to some extent. On the other hand, Aggarwal et al. [27] suggest storing a subset of the set of probabilities in their UFP-Tree using clustering, and also store the highest probability in the cluster to minimize any error in the expected support computation, and thus are able to compute an upper bound on the expected support of an itemset. Another distinction is that whilst Leung et al. [26] build the UF-Tree only for the first two levels and then, for each relevant tree path, enumerate all subsets of each tree path to generate frequent itemsets, Aggarwal et al. [27] build the complete UFP-Tree.

The two different design choices have very different outcomes in terms of the effectiveness of the FP-Tree approach in the probabilistic case. In the corresponding experimental studies, whilst Aggarwal et al. [27] conclude that the UFP-Tree is not effective in terms of the CPU cost and the memory usage, Leung et al. [26] claim that the UF-Tree appears to outperform the U-apriori algorithm.

In addition to extending the FP-Tree to the probabilistic case, Aggarwal et al. [27] consider extending other classical frequent pattern mining algorithms to the probabilistic case as well. Thus, they propose an apriori-based candidate generation

algorithm Uapriori and a hyper-structure based pattern growth algorithm UH-mine. In the classical settings, the H-Mine algorithm uses a hyper-array data structure. Each record in the database is stored as one row in the hyper-array and thus, the hyper-array is easily extended to the probabilistic case by storing one additional row with each record in UH-mine that contains the probabilities. Aggarwal et al. [27] show, with the help of experiments, that UH-mine is the most effective algorithm in terms of CPU cost and memory usage.

This concludes our discussion on mining frequent itemsets from uncertain data using the expected support measure. We now discuss computing frequent itemsets from uncertain data using the probabilistic frequentness measure.

3.6.2 Probabilistic Frequent Itemset Mining

One of the issues with computing the expected support of an itemset in a probabilistic database is that it does not give any information about the confidence in the expected support of an itemset. We explain the potential issue with the help of two examples from Zhang et al. [65]. In the first example, consider a sample database having two tuples $\{\{(a:0.9)(b:0.1)\},\{(c:1.0)\}\}$. Suppose that we are interested in items having expected support equal to 1. The expected support of item a is 0.9 and thus, a is not frequent although there is a 90% probability that the support of a is 1. In another example, again consider a sample database having two tuples $\{\{(a:0.5)\},\{(a:0.5)\}\}$, and suppose that we are again interested in items having expected support equal to 1. The expected support of item a is computed as 1, and a is thus frequent, although the probability that the support of a is 1 is only 75%.

Zhang et al. [65] propose computing $\Pr(Sup(i, D^p) \geq \theta)$, i.e. computing the probability of an item i having support at least θ. In other words, Zhang et al. introduce the concept of confidence in the support of an item in a probabilistic database D^p.

Another important contribution in their work is to consider the x-tuples model which can capture some correlations in the data. Zhang et al. also propose an exact dynamic programming algorithm for computing the probabilistic frequent items, which has quadratic complexity if the x-tuples contain single items, and cubic complexity if the x-tuples contain more than one item. As the computational cost of computing probabilistic frequent items using x-relations model is rather high (cubic), the author also propose a sampling algorithm which has the property that although it can overestimate the support of an item with acceptable error bounds, it will never underestimate it. Zhang et al. also give a pruning technique and show the effectiveness of the proposed exact and sampling algorithms and also the pruning technique with the help of experiments. In summary, the work of Zhang et al. [65] proposes computing the probabilistic frequent items (not itemsets) in a probabilistic database under (a more expressive) x-tuples model.

Bernecker et al. [28] proposed mining probabilistic frequent itemsets from uncertain data. The uncertainty model and the assumption of independence in [28] are similar to [31], i.e. items have existential probabilities and are assumed to exist independently of each other. Bernecker et al. motivate the computation of confidence in the support of an itemset with the help of examples similar to Zhang et al. [65], and then define the probability that the support of an itemset U is exactly k in a probabilistic database D^p as follows:

$$\Pr_{k(U)} = \sum_{(D^* \in PW(D^p)),(Sup(U,D^*)=k)} Pr[D^*],$$

where $Sup(U, D^*)$ is the support of an itemset U in a possible world D^*, and $\Pr_k(U)$ is the probability that the support of U is exactly k in the probabilistic database D^p. Next, they define the *support probability distribution* as the vector $\langle \Pr_0(U), \ldots, \Pr_n(U) \rangle$, and compute the probability that the support of an itemset

U is at least θ, denoted by $\Pr_{\geq\theta}(U)$, as follows:

$$\Pr_{\geq\theta}(U) = \sum_{k=\theta}^{n} \Pr_k(U).$$

An itemset U is a *probabilistic frequent itemset* if $\Pr_{\geq\theta}(U) \geq \tau$, i.e. if the probability that the support of U is at least θ is at least some user-defined confidence threshold[1] τ, $0 < \tau \leq 1$. We now define the probabilistic frequent itemset mining problem.

Definition 3.2 (Probabilistic Frequent Itemset Mining Problem). Given a probabilistic database D^p having n records and two user-defined thresholds, an expected support threshold θ, $0 \leq \theta \leq n$, and a confidence threshold τ, $0 \leq \tau \leq 1$, find all probabilistic frequent itemsets in D^p.

Bernecker et al. give a dynamic programming algorithm to compute probabilistic frequent itemsets which has quadratic time complexity. The proposed optimizations like incremental computation, $0-1$ optimization, and an early pruning technique also do not seem to be particularly effective (the computational complexity of the dynamic programming algorithm does not get any better than $O(\theta n)$). Thus, the performance of the algorithm for harder instances is restricted. This is also backed up by evidence from the experimental study.

Although the dynamic programming algorithm by Bernecker et al. [28] is computationally expensive, it is an exact algorithm. Calders et al. [29] argue that considering the uncertainty model (tuple-level uncertainty) and the assumption of independence between items by Bernecker et al. [28], the probability of the support of an itemset can be approximated with reasonable error bounds. Wang et al. [66] have a similar argument, and in addition to the attribute-level uncertainty model considered by [29], they claim that the approximation scheme proposed by them works for the tuple-level uncertainty model as well, under the assumption that the tuples in a

[1]Note that this confidence threshold is not the same as the one in Section 2.1.2.

probabilistic database are independent of each other. Thus, whilst Calders et al. [29] use a variation of the central limit theorem, Wang et al. [66] use the Poisson binomial distribution to approximate the probability of support of an itemset. In both studies [29, 66], approximation algorithms are given for computing the probability of the support of an itemset. The experimental study by Calders et al. [29] suggests that both for real and synthetic datasets, whilst the mean estimation error does not exceed 0.014% and the maximum estimation error is at most 0.35%, the approximate algorithm is effective in reducing CPU cost relative to the exact dynamic programming algorithm. Similarly, Wang et al. [66] show the effectiveness of their approximation algorithm using *precision* and *recall* [67, Chapter 8] (we define these in Chapter 7), and show that the approximation scheme returns all the relevant results for their considered sets of experiments with a very little error (up to 0.2% in some cases).

As mentioned in Chapter 2, frequent itemset mining is the first step in association rule mining. Sun et al. [25] propose mining association rules from probabilistic frequent itemsets, which they refer to as *probabilistic association rules*. They consider the tuple-level uncertainty model and assume that the tuples are independent of each other in a probabilistic database. Sun et al. propose the p-apriori algorithm and propose optimizations for pruning infrequent patterns, for computing the support probability distribution, and a data structure named *inverted probability list* to improve the algorithm's performance. Sun et al. show the effectiveness of the proposed optimizations with the help of experiments.

This concludes our discussion on frequent itemset mining in uncertain data. We make the following key observations:

(a) All the studies on uncertain frequent itemset mining consider the simpler attribute-level or the tuple-level uncertainty model with the assumption of

independence. The only study that considers a more powerful model, namely x-tuples, finds only frequent items and not the itemsets. It seems that considering a more expressive model makes the frequent itemset computation even harder.

(b) Researchers have focused on two definitions of frequent itemsets (expected support and probabilistic frequentness) and whilst each of these definitions may have its own advantages, it is not clear what might be the 'right' definition (as illustrated for top-k in Section 3.4).

(c) Researchers have been able to adapt the apriori algorithm to the uncertain case both for computing expected and probabilistic frequent itemsets. However, approaches such as pattern growth (FP-Tree) which are considered superior in the classical settings are not straightforward to adapt to the probabilistic case, and it is not clear that they outperform apriori in the probabilistic case as well.

(d) Finding probabilistic frequent itemsets is computationally expensive (quadratic complexity even to find frequent itemsets for the simpler uncertainty models) and researchers have proposed good approximation solutions to the problem.

(e) To the best of our knowledge, there are no studies in literature that evaluate the effectiveness of the probabilistic frequent itemsets mining framework (Wang et al. [66] demonstrate the effectiveness of their approximation algorithm, not the probabilistic frequent itemset mining framework). For example, it would be interesting to contrast the expected and probabilistic frequent itemsets to see if the expensive probabilistic frequentness computation is justified.

3.7 Summary

This concludes our discussion on probabilistic databases. We have focussed on three fundamental tasks that a probabilistic database system performs, namely uncertainty representation and modelling, query evaluation for the chosen uncertainty model and presenting query results to the user, and elaborated on these with the help of real world application examples. We have briefly reviewed representative probabilistic database systems from the literature as well. We have also discussed the semantics of ranking queries (top-k problem) and have seen that although many definitions of top-k have been proposed in literature, a single definition may not suffice for all applications. Similarly, whilst the definitions of frequentness proposed for frequent itemset mining from uncertain data (expected support and probabilistic frequentness) may have their own advantages, it is not clear what might be the 'right' definition. Finally, we have given a detailed account on frequent itemset mining from uncertain data.

Chapter 4

Probabilistic Data Models and Measures

In this chapter a study of probabilistic data models and measures for SPM is presented. We observe that uncertainty in SPM could either be at the attribute level or at the tuple level. We model different kind of uncertainties that could arise in SPM and give motivations using real life examples for our considered uncertainty models. We formally define the uncertainty models we consider and give examples to illustrate the concepts (Section 4.1). We then define the interestingness predicate (Section 4.2) based on the following fundamental problem:

Problem 4.1. *Given a sequence s and a probabilistic database D^p, is s frequent?*

In the framework described by Gunopulos et al. [30], which includes not only SPM but also frequent itemset mining, association rule mining and a host of other database optimization and machine learning problems, the above question is the *interestingness* or *quality* predicate. As noted in [30], given such a predicate as a "black box", one can embed it into a variety of candidate generate-and-test frameworks for finding not only frequent sequential patterns but also maximal frequent sequential

patterns. Many popular classical SPM algorithms such as GSP [38], SPADE [36] or SPAM [34] all fit into this framework, and algorithms such as PrefixSpan [40], which do not explicitly generate candidates, also implicitly evaluate the interestingness predicate.

In this chapter, we define two interestingness predicates and illustrate each of the interestingness predicate we define for the uncertainty models we consider, with the help of examples.

We now discuss probabilistic data models for SPM.

4.1 Probabilistic Data Models

We discuss the kind of uncertainties that could arise in SPM. We first consider the tuple-level uncertain case, and formalize the uncertainty in the tuple, namely *tuple-level uncertainty* (TLU) (Section 4.1.1). We then focus on attribute-level uncertainty, and formalize the uncertainty in the event attribute, namely *event-level uncertainty* (ELU) (Section 4.1.2.1), and uncertainty in the source attribute, namely *source-level uncertainty* (SLU) (Section 4.1.2.2). When modelling the above mentioned uncertainties, some independence assumptions are made which are reasonable for some applications (as we discuss later in this section) but quite a few applications need to capture some basic correlations in data and thus, more expressive models are desirable. We consider another kind of uncertainty that arises in deduplication which we refer to as *source-level uncertainty in a sequence in deduplication* (SLU-D) (Section 4.1.3).

In this section we formally define the notion of a probabilistic database, give its possible world semantics and give examples to illustrate the concepts for the uncertainty models we consider.

TABLE 4.1: A sample TLU database D^p.

eid	e	σ	p
e_1	(a, d, e)	Y	0.4
e_2	(a)	Z	1.0
e_3	(a, d, e)	X	0.6
e_4	(b, c)	Z	0.5
e_5	(b, c)	X	0.3
e_6	(a, b, c)	X	0.7
e_7	(b, c)	Y	0.2
e_8	(c, d, e)	X	1.0
e_9	(a, b, c)	Z	0.3

4.1.1 Tuple-level Uncertainty

Our first model is motivated by the PEEX system (Khoussainova et al. [20]) where employees' movements are tracked in a building using RFID sensors. The stream of tags read by various sensors is stored in a relation `SIGHTING(t, tID, aID)`, which denotes that the RFID tag `tID` was detected by antenna `aID` at time `t`. Since an RFID antenna has only some probability of reading a tag within its range, the PEEX system processes the `SIGHTING` relation to output an *uncertain* higher-level event relation such as `MEETING(time, person1, person2, room, prob)`. An example tuple in `MEETING` could be (103, 'Alice', 'Bob', 435, 0.4), which means that at time 103, PEEX believes that Alice and Bob are having a meeting (event) with probability 0.4 in room 435 (source) (example from [20]); thus, the higher-level event that Alice and Bob are having a meeting only has a probability of 0.4 of having occurred, which shows tuple-level uncertainty. As the RFID tags are read independently of each other by the RFID antenna, we assume that the tuples in `MEETING` are independent of each other. Some formulations of this and similar problems may even exhibit source-level

eid	e	σ	p
e_1	(a, d, e)	Y	0.4
e_2	(a)	Z	1.0
e_3	(a, d, e)	X	0.6
e_4	(b, c)	Z	0.5
e_5	(b, c)	X	0.3
e_6	(a, b, c)	X	0.7
e_7	(b, c)	Y	0.2
e_8	(c, d, e)	X	1.0
e_9	(a, b, c)	Z	0.3

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

FIGURE 4.1: The sample TLU database of Table 4.1 transformed to p-sequences (bottom).

uncertainty. For example, a person may be detected by two different RFID antennae installed in adjacent rooms. In such cases, the source might well be uncertain as well, but we do not consider this case here. We now formalize the idea of a *tuple-level uncertain* (TLU) database.

TLU Database

A TLU database D^p is an ordered list of records $\langle r_1, \ldots, r_n \rangle$ of the form (eid, e, σ, p) where *eid* is an event-id, e is an event, $\sigma \in \mathcal{S}$ is the source and p is the existential probability of the tuple. An example of a TLU database is in Table 4.1.

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 the existential probability of e_j. In examples, we write a p-sequence $\langle(\{a, d\}, 0.4), (\{a, b\}, 0.2)\rangle$ as $\langle(a, d : 0.4)(a, b : 0.2)\rangle$. A TLU 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 σ_i, and contains a list of all the tuples in D^p associated to source σ_i, ordered by *eid*, along with the associated probabilities. Specifically, we generate a p-sequence representation of a TLU database D^p as follows. We initialize D^p_i to an empty list and consider the records $r_1, r_2, \ldots \in D^p$ in that order. When considering the record $r_\ell = (eid_\ell, e_\ell, \sigma_i, p_\ell)$, we append (e_ℓ, p_ℓ) to D^p_i. Figure 4.1 gives an example of this transformation for the TLU database of Table 4.1. Clearly, the p-sequences $D^p_1, \ldots, D^p_m$ provide a dual representation of a TLU database D^p. In a TLU database, in the i-th p-sequence $D^p_i = \langle(e_1, c_1), \ldots, (e_k, c_k)\rangle$, the value c_ℓ represents the existential probability of e_ℓ. As the events occur independently of each other, there are no dependencies among p-sequences in a TLU database.

Possible World Semantics. The possible world semantics of a TLU database D^p is as follows. For each event e_j in a p-sequence D^p_i there are two kinds of possible worlds: one in which e_j occurs and the other where it does not. Let *occur* $= \{x_1, \ldots, x_l\}$, where $1 \leq x_1 < \ldots < x_l \leq k$, be the indices of events that occur in D^*_i. Then $D^*_i = \langle e_{x_1}, \ldots, e_{x_l}\rangle$, and $\Pr(D^*_i) = \prod_{j \in occur} c_j * \prod_{j \notin occur}(1 - c_j)$. We can multiply probabilities because we assume that the events in a p-sequence are stochastically independent. The set of possible worlds for the p-sequence of source σ_i, denoted by $PW(D^p_i)$, is obtained by taking all possible 2^l alternatives for *occur*. For example, the set of possible worlds for each of source X, Y and Z in the sample database of Figure 4.1 is shown in Table 4.2, Table 4.3 and Table 4.4 respectively. We say $PW(D^p) = PW(D^p_1) \times \ldots \times PW(D^p_m)$, where a possible world D^* of D^p is obtained by taking one possible world each from the possible worlds of every source.

TABLE 4.2: The set of possible worlds for source X for the TLU database of Table 4.1.

D^*_X	Possible world	$\Pr(D^*_X)$
$D^*_{X,1}$	$\{(c,d,e)\}$	$(1-0.6)\times(1-0.3)\times(1-0.7)\times 1.0 = 0.084$
$D^*_{X,2}$	$\{(a,d,e)(c,d,e)\}$	$0.6\times(1-0.3)\times(1-0.7)\times 1.0 = 0.126$
$D^*_{X,3}$	$\{(b,c)(c,d,e)\}$	$(1-0.6)\times 0.3\times(1-0.7)\times 1.0 = 0.036$
$D^*_{X,4}$	$\{(a,b,c)(c,d,e)\}$	$(1-0.6)\times(1-0.3)\times 0.7\times 1.0 = 0.196$
$D^*_{X,5}$	$\{(a,d,e)(b,c)(c,d,e)\}$	$0.6\times 0.3\times(1-0.7)\times 1.0 = 0.054$
$D^*_{X,6}$	$\{(a,d,e)(a,b,c)(c,d,e)\}$	$0.6\times(1-0.3)\times 0.7\times 1.0 = 0.294$
$D^*_{X,7}$	$\{(b,c)(a,b,c)(c,d,e)\}$	$(1-0.6)\times 0.3\times 0.7\times 1.0 = 0.084$
$D^*_{X,8}$	$\{(a,d,e)(b,c)(a,b,c)(c,d,e)\}$	$0.6\times 0.3\times 0.7\times 1.0 = 0.126$

TABLE 4.3: The set of possible worlds for source Y for the TLU database of Table 4.1.

D^*_Y	Possible world	$\Pr(D^*_Y)$
$D^*_{Y,1}$	$\langle\rangle$	$(1-0.4)\times(1-0.2) = 0.480$
$D^*_{Y,2}$	$\{(a,d,e)\}$	$0.4\times(1-0.2) = 0.320$
$D^*_{Y,3}$	$\{(b,c)\}$	$(1-0.4)\times 0.2 = 0.120$
$D^*_{Y,4}$	$\{(a,d,e)(b,c)\}$	$0.4\times 0.2 = 0.080$

For any $D^* \in PW(D^p)$ such that $D^* = (D^*_1, \ldots, D^*_m)$, the probability of D^* is given by $\Pr[D^*] = \prod_{i=1}^{m} \Pr(D^*_i)$, as the p-sequences of all sources are mutually independent. For example, one such world D^*_1 is obtained by selecting the possible world $D^*_{X,1}$ from $PW(D^p_X)$, $D^*_{Y,1}$ from $PW(D^p_Y)$ and $D^*_{Z,1}$ from $PW(D^p_Z)$, and the probability of D^*_1 is the product of probabilities of all the possible worlds in it, $\Pr[D^*_1] = 0.084 \times 0.480 \times 0.350 = 0.014$. A demonstration for the complete set of possible worlds for the TLU database of Table 4.1 is in Table 4.5.

Summary. This concludes our discussion of the TLU model. Each tuple in a TLU database has an existential probability and the tuples are assumed to be independent of each other. Thus, whilst the TLU model can be used for modelling uncertainties

TABLE 4.4: The set of possible worlds for source Z for the TLU database of Table 4.1.

D_Z^*	Possible world	$\Pr(D_Z^*)$
$D_{Z,1}^*$	$\{(a)\}$	$1.0 \times (1-0.5) \times (1-0.3) = 0.350$
$D_{Z,2}^*$	$\{(a)(b,c)\}$	$1.0 \times 0.5 \times (1-0.3) = 0.350$
$D_{Z,3}^*$	$\{(a)(a,b,c)\}$	$1.0 \times (1-0.5) \times 0.3 = 0.150$
$D_{Z,4}^*$	$\{(a)(b,c)(a,b,c)\}$	$1.0 \times 0.5 \times 0.3 = 0.150$

TABLE 4.5: The complete set of possible worlds for the TLU database of Table 4.1.

D^*	D_X^*		D_Y^*		D_Z^*		$\Pr(D^*)$
D_1^*	$D_{X,1}^*$	$\{(c,d,e)\}$	$D_{Y,1}^*$	$\{\langle\rangle\}$	$D_{Z,1}^*$	$\{(a)\}$	0.014
D_2^*	$D_{X,1}^*$	$\{(c,d,e)\}$	$D_{Y,1}^*$	$\{\langle\rangle\}$	$D_{Z,2}^*$	$\{(a)(b,c)\}$	0.014
$\vdots$		$\vdots$		$\vdots$		$\vdots$	$\vdots$
D_4^*	$D_{X,1}^*$	$\{(c,d,e)\}$	$D_{Y,1}^*$	$\{\langle\rangle\}$	$D_{Z,4}^*$	$\{(a)(b,c)$ $(a,b,c)\}$	0.006
D_5^*	$D_{X,1}^*$	$\{(c,d,e)\}$	$D_{Y,2}^*$	$\{(a,d,e)\}$	$D_{Z,1}^*$	$\{(a)\}$	0.009
D_6^*	$D_{X,1}^*$	$\{(c,d,e)\}$	$D_{Y,2}^*$	$\{(a,d,e)\}$	$D_{Z,2}^*$	$\{(a)(b,c)\}$	0.009
$\vdots$		$\vdots$		$\vdots$		$\vdots$	$\vdots$
$\vdots$		$\vdots$		$\vdots$		$\vdots$	$\vdots$
D_{16}^*	$D_{X,1}^*$	$\{(c,d,e)\}$	$D_{Y,4}^*$	$\{(a,d,e)$ $(b,c)\}$	$D_{Z,4}^*$	$\{(a)(b,c)$ $(a,b,c)\}$	0.001
D_{17}^*	$D_{X,2}^*$	$\{(a,d,e)(c,d,e)\}$	$D_{Y,1}^*$	$\{\langle\rangle\}$	$D_{Z,1}^*$	$\{(a)\}$	0.021
D_{18}^*	$D_{X,2}^*$	$\{(a,d,e)(c,d,e)\}$	$D_{Y,1}^*$	$\{\langle\rangle\}$	$D_{Z,2}^*$	$\{(a)(b,c)\}$	0.021
$\vdots$		$\vdots$		$\vdots$		$\vdots$	$\vdots$
$\vdots$		$\vdots$		$\vdots$		$\vdots$	$\vdots$
$\vdots$		$\vdots$		$\vdots$		$\vdots$	$\vdots$
D_{128}^*	$D_{X,8}^*$	$\{(a,d,e)(b,c)$ $(a,b,c)(c,d,e)\}$	$D_{Y,4}^*$	$\{(a,d,e)$ $(b,c)\}$	$D_{Z,4}^*$	$\{(a)(b,c)$ $(a,b,c)\}$	0.001

in applications similar to PEEX, it is the simplest of the uncertainty models and is considered a basic model.

We now discuss attribute-level uncertainty in SPM.

4.1.2 Attribute-level Uncertainty

We first consider uncertainty in the event attribute.

4.1.2.1 Event-level Uncertainty

This model is motivated by the following scenarios:

(a) a logged-in user (source) enters search terms into a search engine (events). The search terms or queries can be disambiguated in many different ways. For example, a search term 'Tiger' could potentially be disambiguated as $\{(\text{Animal}, 0.4), (\text{Sports Personality}, 0.3), (\text{Insurance}, 0.2), \ldots\}$.

(b) recall the kind of uncertainty that arises in biological sequences due to mutation (Section 2.5.3), that is, when an observed value D (event) in a sequence (source) could be any of N or D. This kind of uncertainty can also be modelled as ELU. For example, the true value of an observed value D could be $\{(\text{D}, 0.8), (\text{N}, 0.2)\}$.

In such situations, although the source (user/sequence-id) is known, there is uncertainty about the event (search term/observed value) as only one of the many alternatives might be true. Thus, this formulation shows attribute-level uncertainty in the event attribute [24]. We now formalize the notion of an *event-level uncertain* (ELU) database.

TABLE 4.6: A sample ELU database D^p.

eid	E	σ
e_1	$\{(d:1.0)\}$	Z
e_2	$\{(a:0.4),(a,b,d:0.5),(b,c:0.1)\}$	Y
e_3	$\{(a:0.5),(a,b:0.3),(b,c:0.2)\}$	X
e_4	$\{(e:1.0)\}$	Z
e_5	$\{(d:1.0)\}$	X
e_6	$\{(e:1.0)\}$	Y
e_7	$\{(b,c:0.6),(c,d:0.4)\}$	X
e_8	$\{(f,g:1.0)\}$	Y

ELU Database

An ELU database D^p is an ordered list of records $\langle r_1, \ldots, r_n \rangle$ of the form (eid, E, σ) where *eid* is an event-id, E is a set that contains pairs of the form (e, c), where e is an event and $0 < c \leq 1$ is the associated confidence value that the event e is the actual event, and σ is the associated source; we assume $\sum_{(e,c)\in E} c = 1$. An example of an ELU database is in Table 4.6.

p-sequence. In an ELU database, a p-sequence is a sequence of sets of (e, c) pairs in contrast with a TLU database, where it is a sequence of (e, c) pairs. Formally, a p-sequence is a sequence of the form $\langle\{(e_{(1,1)}, c_{(1,1)}), (e_{(1,2)}, c_{(1,2)}), \ldots, (e_{(1,j_1)}, c_{(1,j_1)})\}$ $\ldots\{(e_{(k,1)}, c_{(k,1)}), (e_{(k,2)}, c_{(k,2)}) \ldots, (e_{(k,j_k)}, c_{(k,j_k)})\}\rangle$, where $e_{(\ell,i_\ell)}$ is the i-th event in the ℓ-th set of events and $c_{(\ell,i_\ell)}$ is the associated confidence value. Further, we assume that each set of events consists of only a constant number of events. An ELU database is transformed to p-sequences similar to classical SPM that is all the probabilistic sets of events associated with a single source are listed as a p-sequence of sets of events ordered by a time-stamp. Thus, the p-sequences in an ELU database

eid	E	σ
e_1	$\{(d:1.0)\}$	Z
e_2	$\{(a:0.4),(a,b,d:0.5),(b,c:0.1)\}$	Y
e_3	$\{(a:0.5),(a,b:0.3),(b,c:0.2)\}$	X
e_4	$\{(e:1.0)\}$	Z
e_5	$\{(d:1.0)\}$	X
e_6	$\{(e:1.0)\}$	Y
e_7	$\{(b,c:0.6),(c,d:0.4)\}$	X
e_8	$\{(f,g:1.0)\}$	Y

source	p-sequence
D_X^p	$\{(a:0.5),(a,b:0.3),(b,c:0.2)\}\{(d:1.0)\}\{(b,c:0.6),(c,d:0.4)\}$
D_Y^p	$\{(a:0.4),(a,b,d:0.5),(b,c:0.1)\}\{(e:1.0)\}\{(f,g:1.0)\}$
D_Z^p	$\{(d:1.0)\}\{(e:1.0)\}$

FIGURE 4.2: The ELU database of Table 4.6 transformed to p-sequences (bottom).

have attribute-level uncertainty. However, as in the TLU case, the sets of events in a p-sequence or the p-sequences itself are independent of each other. An example of the transformation of the ELU database of Table 4.6 is in Figure 4.2.

Possible World Semantics. A possible world D^* of an ELU database D^p is generated by taking each record r_i in turn, and selecting one of the possible events $e_i \in E_i$. Thus every record $r_i = (eid_i, E_i, \sigma_i) \in D^p$ takes the form $r_i' = (eid_i, e_i, \sigma_i)$ for some $e_i \in E_i$, in D^*. By enumerating all such possible combinations, we get the complete set of possible worlds. Assuming that the uncertain set of events associated with each record r_i in D^p are stochastically independent of each other, the probability of a possible world D^* is $\Pr[D^*] = \prod_{i=1}^{n} \Pr_{E_i}[e_i]$. For example, a possible world D^* for the ELU database of Table 4.6 can be generated by selecting the events (a), (a), (b,c) from e_2, e_3 and e_7 with probabilities $0.4, 0.5$ and 0.6 respectively, and $\Pr[D^*] =$

TABLE 4.7: The complete set of possible worlds for the sample ELU database of Table 4.6 along with their probabilities.

D^*	X	Y	Z	$\Pr(D^*)$
D_1^*	$(a)(d)(b,c)$	$(a)(e)(f,g)$	$(d)(e)$	0.120
D_2^*	$(a)(d)(b,c)$	$(a,b,d)(e)(f,g)$	$(d)(e)$	0.150
D_3^*	$(a)(d)(b,c)$	$(b,c)(e)(f,g)$	$(d)(e)$	0.030
D_4^*	$(a)(d)(c,d)$	$(a)(e)(f,g)$	$(d)(e)$	0.080
D_5^*	$(a)(d)(c,d)$	$(a,b,d)(e)(f,g)$	$(d)(e)$	0.100
D_6^*	$(a)(d)(c,d)$	$(b,c)(e)(f,g)$	$(d)(e)$	0.020
D_7^*	$(a,b)(d)(b,c)$	$(a)(e)(f,g)$	$(d)(e)$	0.072
D_8^*	$(a,b)(d)(b,c)$	$(a,b,d)(e)(f,g)$	$(d)(e)$	0.090
D_9^*	$(a,b)(d)(b,c)$	$(b,c)(e)(f,g)$	$(d)(e)$	0.018
D_{10}^*	$(a,b)(d)(c,d)$	$(a)(e)(f,g)$	$(d)(e)$	0.048
D_{11}^*	$(a,b)(d)(c,d)$	$(a,b,d)(e)(f,g)$	$(d)(e)$	0.060
D_{12}^*	$(a,b)(d)(c,d)$	$(b,c)(e)(f,g)$	$(d)(e)$	0.012
D_{13}^*	$(b,c)(d)(b,c)$	$(a)(e)(f,g)$	$(d)(e)$	0.048
D_{14}^*	$(b,c)(d)(b,c)$	$(a,b,d)(e)(f,g)$	$(d)(e)$	0.060
D_{15}^*	$(b,c)(d)(b,c)$	$(b,c)(e)(f,g)$	$(d)(e)$	0.012
D_{16}^*	$(b,c)(d)(c,d)$	$(a)(e)(f,g)$	$(d)(e)$	0.032
D_{17}^*	$(b,c)(d)(c,d)$	$(a,b,d)(e)(f,g)$	$(d)(e)$	0.040
D_{18}^*	$(b,c)(d)(c,d)$	$(b,c)(e)(f,g)$	$(d)(e)$	0.008

$0.5 \times 1.0 \times 0.6 \times 0.4 \times 1.0 \times 1.0 \times 1.0 \times 1.0 = 0.120$. The complete set of possible worlds for database of Table 4.6 is in Table 4.7.

Summary. This concludes our discussion on the ELU model. Our proposed ELU model is similar to the TLU model as both: (1) model uncertainty in the event attribute and (2) the tuples are assumed to be independent of each other. However, as we assume that $\sum_{(e,c)\in E} c = 1$, a tuple is always present in a possible world in the ELU model although the event attribute in a tuple may take different values; whereas tuples have existential probabilities in the TLU model. If the above condition is modified to $\sum_{(e,c)\in E} c \leq 1$, then an event can not only take different values

but it may also not appear in a possible world (which is similar to the TLU model). Thus, our ELU model can be seen to some extent as a generalization of the TLU model.

We now discuss the uncertainty in the source attribute in a tuple.

4.1.2.2 Source-level Uncertainty

Our SLU model is motivated by the following situations:

(a) a customer (source) purchases some items (event) from a superstore, and provides identity information, e.g. by filling a form. The same customer may fill a *new* form in a subsequent visit and thus, multiple matches may emerge in the customer database as the customer's details may be incomplete or incorrect.

(b) a vehicle (source) is identified by a camera (event) using methods such as automatic number plate recognition, 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 one of the vehicles from {(Y81 UV: 0.6)(YB1 UV: 0.3)(Y81 UU:0.1)} was sighted at time 103 at location p. 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 such technology is the UK police automatic number

TABLE 4.8: A sample SLU database D^p.

eid	e	W
e_1	(a, d, e)	$(X : 0.6)(Y : 0.4)$
e_2	(a)	$(Z : 1.0)$
e_3	(b, c)	$(X : 0.3)(Y : 0.2)(Z : 0.5)$
e_4	(a, b, c)	$(X : 0.7)(Z : 0.3)$
e_5	(c, d, e)	$(X : 1.0)$

plate recognition database [68], and studies suggest that even the most advanced automatic number plate recognition systems have only upto 90% accuracy even under most suitable weather conditions [69]. 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 [24]. We now formalize a *source-level uncertain* (SLU) database.

SLU Database

An SLU database D^p is an ordered list of records $\langle r_1, \ldots, r_n \rangle$ 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 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; we assume $\sum_{(\sigma,c)\in W} c = 1$, and that the distributions associated with each record r_i in D^p are stochastically independent. An example of an SLU database is in Table 4.8.

p-sequence. A p-sequence in an SLU database is similar to a p-sequence in a TLU database although, the p-sequence generation details for an SLU database are

eid	*e*	*W*
e_1	(a, d, e)	$(X : 0.6)(Y : 0.4)$
e_2	(a)	$(Z : 1.0)$
e_3	(b, c)	$(X : 0.3)(Y : 0.2)(Z : 0.5)$
e_4	(a, b, c)	$(X : 0.7)(Z : 0.3)$
e_5	(c, d, e)	$(X : 1.0)$

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

FIGURE 4.3: The SLU database of Table 4.8 transformed to p-sequences (bottom). Note that in the p-sequence representation, events like e_1 (marked with †) can only be associated with one of the sources X and Y in any possible world.

slightly different. Specifically, we generate a p-sequence representation of an SLU database D^p as follows. We initialize D^p_i to an empty list and consider the records $r_1, r_2, \ldots \in D^p$ in that order. When considering the record $r_\ell = (eid_\ell, e_\ell, W_\ell)$, if W_ℓ associates e_ℓ with source σ_i with confidence $c > 0$, we append (e_ℓ, c) to D^p_i. Figure 4.3 gives an example of this transformation for the SLU database of Table 4.8. The assumption of independence of the distributions associated with different records in an SLU database implies that the event that e_ℓ is associated with source σ_i is independent of the event that $e_{\ell'}$ is associated with source σ_i, for any pair of indices $1 \leq \ell, \ell' \leq k$, $\ell \neq \ell'$. Thus, whilst the p-sequences have tuple-level uncertainty in an SLU database similar to a TLU database, the p-sequences corresponding to different sources in an SLU database are *not* independent (as illustrated in Figure 4.3) and thus, one may view an SLU database as a collection of p-sequences with dependencies in the form of x-tuples [3].

TABLE 4.9: The complete set of possible worlds for the database of Table 4.8 along with their probabilities.

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

Possible World Semantics. A possible world D^* of an SLU database D^p is generated by taking each event e_i in turn, and assigning it to one of the possible sources $\sigma_i \in W_i$, where $\sigma_i \in \mathcal{S}$. 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. Assuming that the distributions associated with each record r_i in D^p are stochastically independent, the probability of a possible world D^* is $\Pr[D^*] = \prod_{i=1}^{n} \Pr_{W_i}[\sigma_i]$. For example, a possible world D^* for the database of Table 4.8 can be generated by associating event e_2 to source Z with probability 1.0, and the rest of the events that is events e_1, e_3, e_4 and e_5 to source X with probabilities $0.6, 0.3, 0.7$ and 1.0 respectively, and $\Pr[D^*] = 0.6 \times 1.0 \times 0.3 \times 0.7 \times 1.0 = 0.126$. The complete set of possible worlds for database of Table 4.8 is in Table 4.9.

Summary. This concludes our discussion on the SLU model. So far, our discussion on uncertainty models for SPM has focussed either on tuple-level uncertainty (TLU)

or attribute-level uncertainty in a tuple (ELU and SLU). As both ELU and TLU (in some respect) model uncertainty in an attribute (event) and SLU is also about uncertainty in an attribute (source), it might suggest that all these three models are similar. However, this is not entirely the case as SPM is focussed on the source attribute (recall the notion of a source sequence from Chapter 2). Thus, whilst our TLU and ELU models are least constrained, as they assume independence among events in a p-sequence and also among different p-sequences, our SLU model is slightly constrained as it introduces dependencies among p-sequences.

In the following section, we model the uncertainty in the source attribute that arises in deduplication (SLU-D).

4.1.3 Uncertainty in Deduplication

We add to the expressiveness of the uncertainty models we propose by introducing a new kind of uncertainty, motivated by deduplication. Thus, our SLU-D model is motivated by a scenario where a customer has registered for a loyalty programme and is identified by a loyalty card etc. on his visit to a superstore. The information about the customer loyalty programme is stored in a database which is uncertain due to deduplication. Thus, when a customer (source) purchases items (event) from a superstore in multiple visits and is identified each time (deterministically) by a loyalty card etc.; multiple matches may emerge in the customer database as the customer database itself is uncertain as a result of deduplication or cleaning [2].

In such situations, whilst the customer (source) and the associated transactions (sequence of events) are recorded deterministically (which means that both the source and the source sequence are certain), the source attribute in a source sequence exhibits uncertainty due to deduplication [2]. Note that the main difference in SLU and SLU-D models is that all uncertainties associated to a source are resolved the

Activity		
eid	*e*	*sid*
e_1	(a)	sid_1
e_2	(b)	sid_2
e_3	(c)	sid_1
e_4	(d)	sid_2

Mapping	
sid	*W*
sid_1	$(X : 0.7)(Y : 0.3)$
sid_2	$(Y : 0.6)(Z : 0.4)$

FIGURE 4.4: A sample SLU-D database D^p.

same way in the SLU-D model (which we discuss later). We now formalize the notion of a *source-level uncertain deduplicated* (SLU-D) database.

SLU-D Database

An SLU-D database D^p is a set of two relations, the *activity* relation $\mathcal{A}$ and the *mapping* relation $\mathcal{M}$. The activity relation $\mathcal{A}$ is an ordered list of records $\langle a_1, a_2, \ldots, a_n \rangle$ where each record is of the form (eid, e, sid) where eid is an event-id, e is an event and sid is the source-id (records are ordered by eid). Unlike an SLU database, we assume that the sids are certain although, the mapping from sid to a source σ is uncertain and is captured by the mapping relation $\mathcal{M}$. The mapping relation $\mathcal{M}$ is also a list of records $\langle m_1, m_2, \ldots, m_l \rangle$ of the form (sid, W) where every sid is associated with a distribution W over $\mathcal{S}$. The distribution W contains pairs of the form $(\sigma : c)$, where $\sigma \in \mathcal{S}$, and $0 < c \leq 1$ is the confidence value that the corresponding

TABLE 4.10: The SLU-D database of Figure 4.4 transformed to *sid*-sequences.

sid	*sid*-sequence	W
sid_1	$\langle\{e_1 : (a)\}\{e_3 : (c)\}\rangle$	$(X : 0.7)(Y : 0.3)$
sid_2	$\langle\{e_2 : (b)\}\{e_4 : (d)\}\rangle$	$(Y : 0.6)(Z : 0.4)$

TABLE 4.11: The SLU-D database of Table 4.10 transformed to p-sequences. Note that the events like e_1 and e_3 (marked with †) can only be associated to one of the sources X and Y in any possible world. Further, events like e_2 and e_4 (marked with $\star$) will either both be associated to a source in a possible world or otherwise.

source	p-sequence
X	$\{e_1, (a) : 0.7\}^\dagger\{e_3, (c) : 0.7\}^\dagger$
Y	$\{e_1, (a) : 0.3\}^\dagger\{e_2, (b) : 0.6\}^\star\{e_3, (c) : 0.3\}^\dagger\{e_4, (d) : 0.6\}^\star$
Z	$\{e_2, (b) : 0.4\}\{e_4, (d) : 0.4\}$

sid is associated with the source σ. We assume that the distributions associated with different *sid*s in $\mathcal{M}$ are independent of each other and that $\sum_{(\sigma:c)\in W} c = 1$.

sid-sequence. An *sid*-sequence is analogous to a source sequence in classical SPM, as all the events associated with an *sid* are listed as a single sequence of events (ordered by a time-stamp). As already mentioned that the mapping from an *sid* to a source σ is uncertain, an SLU-D database can be seen as a collection of *sid*-sequences where there is uncertainty about an *sid*-sequence being associated to a source σ. The SLU-D database of Figure 4.4 is transformed to a collection of *sid*-sequences in Table 4.10.

p-sequence. The *sid*-sequences can also be transformed to a representation similar to the p-sequences in a way similar to the SLU case (Section 4.1.2.2). The p-sequence representation for the *sid*-sequences in Table 4.10 is in Table 4.11. Note that in the p-sequence representation in Table 4.11, events like e_1 and e_3 (marked with †) can

TABLE 4.12: The complete set of possible worlds for the database of Table 4.10 along with their probabilities.

D^*	X	Y	Z	$\Pr(D^*)$
D_1^*	$\{e_1:(a)\}\{e_3:(c)\}$	$\{e_2:(b)\}\{e_4:(d)\}$	$\langle\rangle$	0.420
D_2^*	$\{e_1:(a)\}\{e_3:(c)\}$	$\langle\rangle$	$\{e_2:(b)\}\{e_4:(d)\}$	0.280
D_3^*	$\langle\rangle$	$\{e_1:(a)\}\{e_2:(b)\}$ $\{e_3:(c)\}\{e_4:(d)\}$	$\langle\rangle$	0.180
D_4^*	$\langle\rangle$	$\{e_1:(a)\}\{e_3:(c)\}$	$\{e_2:(b)\}\{e_4:(d)\}$	0.120

only be associated with one of the sources X and Y in a possible world. Further, the events in an *sid*-sequence, e.g. the events e_2 and e_4 in sid_2 (marked with $\star$ in Table 4.11) will all either be associated to a source or otherwise in a possible world. Thus, in an SLU-D database, there are dependencies among events in a p-sequence as well as among different p-sequences.

Possible World Semantics. The possible world semantics of an SLU-D database D^p is as follows. A possible world D^* of D^p is generated by taking each *sid*-sequence sid_i in turn, and assigning it to one of the possible sources $\sigma_i \in W_i$. Thus, all the records associated with sid_i in $\mathcal{A}$ are associated with a single source σ in a possible world D^*. By enumerating all such possible combinations, we get the complete set of possible worlds. We assume that the distributions associated with each *sid* in $\mathcal{M}$ are stochastically independent of each other, which means that mapping of sid_i to source σ_j is independent of $sid_{i'}$, $i' \neq i$, being mapped to source σ_j. The probability of a possible world D^* is computed as $\Pr[D^*] = \prod_{i=1}^{l} \Pr_{W_i}[\sigma_i]$. For example, a possible world D^* of the SLU-D database in Table 4.10 is generated by associating both the *sid*-sequences, namely sid_1 and sid_2, with sources Y. The probability of such a world is $0.3 * 0.6 = 0.18$. The complete set of possible worlds for the SLU-D database of Table 4.10 is in Table 4.12.

Summary. We note that SLU-D is a more expressive model in contrast with TLU or ELU which assume independence among events in a p-sequence as well as among different p-sequence, and in contrast with SLU which assumes independence among events in a p-sequence. An important distinction in SLU and SLU-D models is that whilst in the SLU model the uncertainty in the source attribute is resolved independently for each tuple, in the SLU-D model the uncertainties for all the tuples associated to a specific source (e.g. loyalty card) are resolved the same way. Thus, SLU-D model captures dependencies not only among different p-sequences but also among events in a p-sequence.

Remark 4.2. It appears as if capturing dependencies in an uncertainty model may make uncertain data modelling more realistic, but it can make evaluating the interestingness predicate (defined in Section 4.2) computationally intractable as illustrated in Chapter 5, Sections 5.1.3 and 5.2.2.

Summary. This concludes our discussion on probabilistic data models for SPM. We have shown that different kind of uncertainties could arise in a variety of applications that can be modelled using different uncertainty models. Thus, models like TLU or ELU are least constrained (simple) as they are based on some independence assumptions whereas, models like SLU or SLU-D are more expressive as they capture some basic correlations in data.

We now define the interestingness predicate.

4.2 The Interestingness Predicate

The interestingness predicate is usually defined based on some interestingness measure or frequency criterion; we define two interestingness measures namely *expected*

support and *probabilistic frequentness* for probabilistic SPM and customize the interestingness predicate from Problem 4.1 accordingly. We use possible worlds semantics for the purpose and give examples for evaluating the interestingness predicate for each of the uncertainty models proposed in Section 4.1.

4.2.1 Expected Support

We define the expected support of a sequence s in a probabilistic database D^p using possible world semantics. As every possible world D^* is a (deterministic) database, the support of s in D^*, denoted by $Sup(s, D^*)$, follows directly from Equation 2.2. We then define the expected support of a sequence s in a probabilistic database D^p as follows:

$$ES(s, D^p) = \sum_{D^* \in PW(D^p)} \Pr[D^*] * Sup(s, D^*). \tag{4.1}$$

We now give examples of computing the expected support of a sequence $\langle(a)(b)\rangle$ for each of the sample probabilistic databases in Section 4.1 using the possible worlds. In our examples, Tables 4.13, 4.14, 4.15 and 4.16, show the computing of the expected support of $\langle(a)(b)\rangle$, for the sample probabilistic databases of Tables 4.1, 4.6, 4.8 and 4.10, using the possible worlds from Tables 4.5, 4.7, 4.9 and 4.12, respectively. Note that the TLU database of Table 4.1 and the SLU database of Table 4.8, which have the same p-sequence form as shown in Figures 4.1 and 4.3, have the same expected support for $\langle(a)(b)\rangle$, even though the possible worlds (see Tables 4.5 and 4.9) are very different.

Next, we define the notion of an *expected frequent sequence*:

Definition 4.3 (**Expected Frequent Sequence).** Given a sequence s, a probabilistic database D^p having m sources and a user-defined expected support threshold θ, $1 \leq \theta \leq m$, s is an expected frequent sequence if $ES(s, D^p) \geq \theta$.

TABLE 4.13: Computing the expected support of a sequence $\langle(a)(b)\rangle$ using possible worlds (Table 4.5) for the sample TLU database of Table 4.1.

		D_1^*		D_2^*		D_3^*		·····		D_{127}^*		D_{128}^*
$\Pr(D^*)$		0.014		0.014		0.006		·····		0.001		0.001
$Sup(\langle(a)(b)\rangle, D^*)$		0		1		1		·····		3		3
$ES(\langle(a)(b)\rangle, D^p)$	=	0.014		0.014		0.006				0.001		0.001
		×	+	×	+	×	+	·····	+	×	+	×
		0		1		1				3		3
	=	0.000	+	0.014	+	0.006	+	·····	+	0.003	+	0.003
	=	1.204										

TABLE 4.14: Computing the expected support of a sequence $\langle(a)(b)\rangle$ using possible worlds (Table 4.7) for the sample ELU database of Table 4.6.

		D_1^*		D_2^*		D_3^*		·····		D_{17}^*		D_{18}^*
$\Pr(D^*)$		0.120		0.150		0.030		·····		0.040		0.008
$Sup(\langle(a)(b)\rangle, D^*)$		1		1		1		·····		0		0
$ES(\langle(a)(b)\rangle, D^p)$	=	0.120		0.150		0.030				0.040		0.008
		×	+	×	+	×	+	·····	+	×	+	×
		1		1		1				0		0
	=	0.120	+	0.150	+	0.030	+	·····	+	0.000	+	0.000
	=	0.480										

We now formalize the computational task of finding all expected frequent sequences in a probabilistic database D^p:

Definition 4.4 (**Expected Frequent Sequence Mining Problem).** Given a probabilistic database D^p having m sources and a user-defined expected support threshold θ, $1 \leq \theta \leq m$, find all expected frequent sequences in D^p.

TABLE 4.15: Computing the expected support of a sequence $\langle(a)(b)\rangle$ using possible worlds (Table 4.9) for the sample SLU database of Table 4.8.

		D_1^*		D_2^*		D_3^*		·····		D_{11}^*		D_{12}^*
$\Pr(D^*)$		0.126		0.054		0.084		·····		0.140		0.060
$Sup(\langle(a)(b)\rangle, D^*)$		1		2		1		·····		2		1
$ES(\langle(a)(b)\rangle, D^p)$	=	0.126		0.054		0.084				0.140		0.060
		×	+	×	+	×	+	·····	+	×	+	×
		1		2		1				2		1
	=	0.126	+	0.108	+	0.084	+	·····	+	0.280	+	0.060
	=	1.204										

TABLE 4.16: Computing the expected support of a sequence $\langle(a)(b)\rangle$ using possible worlds (Table 4.12) for the sample SLU-D database of Table 4.10.

		D_1^*		D_2^*		D_3^*		D_4^*
$\Pr(D^*)$		0.420		0.280		0.180		0.120
$Sup(\langle(a)(b)\rangle, D^*)$		0		0		1		0
$ES(\langle(a)(b)\rangle, D^p)$	=	0.420		0.280		0.180		0.120
		×	+	×	+	×	+	×
		0		0		1		0
	=	0.000	+	0.000	+	0.180	+	0.000
	=	0.180						

4.2.2 Probabilistic Frequentness

Recall from Section 3.6.2 that one of the criticisms of expected support is that the expectation of a random variable does not provide confidence bounds that the support of a sequence is high [28]. We now define the notion of *probabilistic frequentness* of a sequence. We first give a few notations.

Given a probabilistic database D^p and its set of possible worlds $PW(D^p)$, the *support probability* for a sequence s and a support value k is denoted by:

$$\Pr_k(s) = \sum_{D^* \in PW(D^p), (Sup(s,D^*)=k)} \Pr(D^*), \tag{4.2}$$

where $Sup(s, D^*)$ is the support of s in D^*. In other words, $\Pr_k(s)$ is the probability that the support of s is exactly k in the probabilistic database D^p. Next define the *support probability distribution* as the vector $\langle \Pr_0(s), \ldots, \Pr_m(s) \rangle$. The support probability distributions for a sequence $s = \langle (a)(b) \rangle$, for each of the sample probabilistic database in Section 4.1 are shown in Figure 4.5. Observe that the support probability distribution for the SLU database of Table 4.8 is very different from the TLU database of Table 4.1, although the p-sequences are the same (Figures 4.3 and 4.1). For example, for the TLU database of Table 4.1, $\Pr_3(s) = 0.026$, but in the SLU database of 4.8, $\Pr_3(s) = 0$, as no such world exists where all three sources support s (see Table 4.9). Finally, the probability that the support of a sequence s is at least θ, denote by $\Pr_{\geq\theta}(s)$, is computed as follows:

$$\Pr_{\geq\theta}(s) = \sum_{k=\theta}^{m} \Pr_k(s) \tag{4.3}$$

Next, we define the notion of a *probabilistic frequent sequence*:

Definition 4.5 (Probabilistic Frequent Sequence). Given a sequence s, a probabilistic database D^p having m sources and two user-specified thresholds, a support threshold θ, $1 \leq \theta \leq m$, and a confidence threshold $\tau \in (0, 1]$, s is a probabilistic frequent sequence if $\Pr_{\geq\theta}(s) \geq \tau$ or alternatively, s is a probabilistic frequent sequence if it has a probability of at least τ of having support at least θ in the probabilistic database D^p.

We now formalize the computational task of finding all probabilistic frequent sequences in a probabilistic database D^p:

for the TLU database of Table 4.1				
	Number of sources			
	0	1	2	3
Support probability	0.024	0.855	0.095	0.026

for the ELU database of Table 4.6				
	Number of sources			
	0	1	2	3
Support probability	0.520	0.480	0.000	0.000

for the SLU database of Table 4.8				
	Number of sources			
	0	1	2	3
Support probability	0.084	0.628	0.288	0.000

for the SLU-D database of Table 4.10				
	Number of sources			
	0	1	2	3
Support probability	0.820	0.180	0.000	0.000

FIGURE 4.5: The support probability distributions for a sequence $\langle(a)(b)\rangle$ for each of the sample probabilistic database in Section 4.1.

Definition 4.6 (Probabilistic Frequent Sequence Mining Problem). Given a probabilistic database D^p having m sources and two user-specified thresholds, a support threshold θ, $1 \leq \theta \leq m$, and a confidence threshold $\tau \in (0, 1]$, find all probabilistic frequent sequences in D^p.

Observe that the support probability distribution gives far more detailed information than expected support; from the support probability distribution of a sequence

one can easily compute not only the expected support, but also the variance and higher moments. For example, the expected support of a sequence $\langle(a)(b)\rangle$ in the sample SLU database of Table 4.8 using the support probability distribution in Figure 4.5 can be computed as $0 \times 0.084 + 1 \times 0.628 + 2 \times 0.288 + 3 \times 0.0 = 1.204$, which is the same as the value computed in Table 4.15.

4.3 Summary

We have discussed probabilistic data models for the SPM problem motivated by real life applications. We have considered uncertainty in the tuple, and in the event or the source attribute of a tuple. We have also modelled another kind of uncertainty that could arise in the source attribute in deduplication. Finally, we have defined the interestingness predicate based on two frequency criteria, namely expected support and probabilistic frequentness, and have illustrated the concepts with the help of examples.

Chapter 5

Computational Complexity of Evaluating the Interestingness Predicate

We consider the problem of computing frequent sequences in a probabilistic database for the uncertainty models and the definitions of interestingness proposed in Chapter 4. We show that different formalizations of the probabilistic SPM problem can lead to very different outcomes from a complexity theoretic viewpoint. We focus on evaluating the interestingness predicate (Section 4.2), which when specialized to probabilistic SPM, and to the definitions of expected frequent sequence (Section 4.2.1) and probabilistic frequent sequence (Section 4.2.2), yield the following problems.

Problem 5.1. *Given a probabilistic database D^p, a sequence s and an expected support threshold θ, is $ES(s, D^p) \geq \theta$?*

Problem 5.2. *Given a probabilistic database D^p, a sequence s, an expected support threshold θ and a confidence threshold τ, is $\Pr(Sup(s, D^p) \geq \theta) \geq \tau$?*

We now discuss Problems 5.1 and 5.2 in the context of each of the uncertainty models in Chapter 4, namely TLU, ELU, SLU and SLU-D.

We first give a few notations. We denote the number of sources by m (as in Chapter 4), the number of events in the p-sequence of the i-th source by N_i, the total size of all p-sequences by $N = \sum_{i=1}^{m} N_i$, and the number of elements in s by k. We assume that an element consists of at most a constant number of items. Further, in an ELU database, we denote the number of (sets of) events in the p-sequence of the i-th source by N_i as well for convenience' sake.

5.1 Expected Support Computation

We now discuss Problem 5.1, that is, we focus on evaluating the interestingness predicate under the expected support measure for the uncertainty models we propose. As mentioned in Chapter 4, there are potentially an exponential number of possible worlds, so it is infeasible to evaluate the interestingness predicate directly by computing the expected support of a sequence using Equation 4.1. We show that whilst we can evaluate the interestingness predicate efficiently for the TLU, ELU and SLU-models, it is computationally intractable to evaluate interestingness predicate for the SLU-D model. We first show that we can process p-sequences independently for the purpose of expected support computation even if there are dependencies among p-sequences (e.g. in the case of an SLU database) due to the principle of linearity of expectation.

Theorem 5.3. *Given a probabilistic database D^p in the form of p-sequences of events and a sequence s, we can compute the expected support of s in D^p by computing the source support probability for each source σ_i independently, that is $ES(s, D^p) = \sum_{i=1}^{m} \Pr(s \preceq D_i^p)$.*

Proof. Recall from Equation 4.1 that the expected support of a sequence s in a probabilistic database D^p is given by:

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

Now recall from Section 2.2.1 that we defined an indicator variable X_i whose value is 1 if s is a subsequence of the source sequence for source σ_i and 0 otherwise. As the support of s in a possible world D^*, denoted by $Sup(s, D^*)$, is the same as the value $\sum_{i=1}^{m} X_i(s, D^*)$, we can write:

$$\begin{aligned} ES(s, D^p) &= \textstyle\sum_{D^*} \Pr[D^*] * \sum_{i=1}^{m} X_i(s, D^*) \\ &= \textstyle\sum_{i=1}^{m} \sum_{D^*} \Pr[D^*] * X_i(s, D^*) \\ &= \textstyle\sum_{i=1}^{m} \mathrm{E}[X_i(s, D^p)], \end{aligned} \tag{5.1}$$

where E denotes the expected value of a random variable X_i for a sequence s in the probabilistic database D^p. Since X_i is a 0-1 variable,

$$\mathrm{E}[X_i(s, D^p)] = \Pr[s \preceq D_i^p], \tag{5.2}$$

and we calculate the right-hand quantity in Equation 5.2, which we refer to as the *source support probability*, and is the probability that a source σ_i supports a sequence s.

Thus, from Equations 5.1 and 5.2, we show that computing the expected support of a sequence s in a probabilistic database D^p is equivalent to computing the sum of the source support probabilities for each of the p-sequences D_i^p in the probabilistic database D^p, that is:

$$ES(s, D^p) = \textstyle\sum_{i=1}^{m} \Pr[s \preceq D_i^p], \tag{5.3}$$

TABLE 5.1: Computing $\Pr[\langle(a)(b)\rangle \preceq D_X^p]$ using dynamic programming in the sample database of Figure 4.3. In Table 5.1, the value A[2,4] is the probability that the sequence $\langle(a)(b)\rangle$ is supported by source X.

		$(a, d, e : 0.6)$	$(b, c : 0.3)$	$(a, b, c : 0.7)$	$(c, d, e : 1.0)$
	A[0,0] = 1	A[0,1] = 1	A[0,2] = 1	A[0,3] = 1	A[0,4] = 1
$\langle(a)\rangle$	A[1,0] = 0	A[1,1] = 0.4× A[1,0] + 0.6× A[0,0] = 0.6	A[1,2] = 0.6	A[1,3] = 0.3× A[1,2] + 0.7× A[0,2] = 0.72	A[1,4] = 0.72
$\langle(a)(b)\rangle$	A[2,0] = 0	A[2,1] = 0	A[2,2] = 0.7× A[2,1] + 0.3× A[1,1] = 0.18	A[2,3] = 0.3× A[2,2] + 0.7× A[1,2] = 0.474	A[2,4] = 0.474

which proves the theorem.

□

We now discuss computing the source support probability and hence, the expected support of a sequence in a probabilistic database. Recall from Section 4.1 that the p-sequence representation of an ELU database is different from that of an SLU or a TLU database; we first consider computing expected support in an SLU (or a TLU) database, and then consider the ELU case.

5.1.1 TLU/SLU Case

We focus on computing the source support probability in an SLU (or a TLU) database. Although the assumption of independence between events in a p-sequence means that there is no combinatorial explosion arising directly from the exponential number of possible worlds, an important issue is that s may be a subsequence of D_i^p in many different ways. For example, if $s = \langle\underbrace{(a)(a)\ldots(a)}_{k \text{ times}}\rangle$ and $D_i^p = \langle(a : c_1), (a, c_2), \ldots, (a, c_{N_i})\rangle$, then any subset of k positions from D_i^p could be the (sole) basis for the i-th source to support s. Thus, we cannot compute the source support probability naively. We now show that we can compute the source support probability efficiently using dynamic programming.

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 $1 \leq k \leq q$ and $1 \leq \ell \leq r$, $A[k, \ell]$ 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 $A[2, 4]$ in Table 5.1 contains the value $\Pr[\langle (a)(b) \rangle \preceq D_X^p]$. We set $A[0, \ell] = 1$ for all ℓ, $0 \leq \ell \leq r$ and $A[k, 0] = 0$ for all $1 \leq k \leq q$, and compute the other values row-by-row. For $1 \leq k \leq q$ and $1 \leq \ell \leq r$, define:

$$c_{k\ell}^* = \begin{cases} c_\ell & \text{if } s_k \subseteq e_\ell \\ 0 & \text{otherwise} \end{cases} \tag{5.4}$$

The interpretation of Equation 5.4 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:

$$A[k, \ell] = (1 - c_{k\ell}^*) * A[k, \ell - 1] + c_{k\ell}^* * A[k - 1, \ell - 1]. \tag{5.5}$$

Table 5.1 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 Figure 4.3. Similarly, we can compute $\Pr[s \preceq D_Y^p] = 0.08$ and $\Pr[s \preceq D_Z^p] = 0.65$, so the expected support of $\langle (a)(b) \rangle$ in the database of Figure 4.3 is $0.474 + 0.08 + 0.65 = 1.204$. The reason Equation 5.5 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 Equation 5.5) and those where it is (the second term in Equation 5.5). We have therefore shown:

Lemma 5.4. *Given a p-sequence D_i^p and a sequence s, by applying Eq. 5.5 repeatedly, we correctly compute* $\Pr[s \preceq D_i^p]$.

To calculate the expected support of a sequence s in an SLU or a TLU database D^p using Equation 5.3, we can apply Lemma 5.4 to each source in turn, which takes $O(k \cdot \sum_{i=1}^{m} N_i) = O(kN)$ time.

Theorem 5.5. *Given a sequence s and an SLU or a TLU database D^p, we can calculate the expected support of s in D^p and hence evaluate the interestingness predicate, in $O(kN)$ time.*

Proof. Given a sequence $s = \langle s_1, \ldots, s_q \rangle$ and the p-sequence of source σ_i of size N_i, we create a $(q+1) \times (N_i + 1)$ matrix $A[0..q][0..N_i]$ and do the initialisations as in Section 5.1.1. We then compute the entries in the A matrix row-by-row. We assume that the test in Equation 5.4 can be done in constant time. Clearly, the source support probability computation for a single source takes $O(k \cdot N_i)$ time. Thus, we can compute the expected support of a sequence s in an SLU or a TLU database by applying Lemma 5.4 to each source in turn, which takes $O(k \cdot \sum_{i=1}^{m} N_i) = O(kN)$ time and this proves the theorem.

□

We now discuss the ELU case.

5.1.2 ELU Case

Recall that in an ELU database, a p-sequence is of the form $\langle \{(e_{(1,1)}, c_{(1,1)}), (e_{(1,2)}, c_{(1,2)}), \ldots, (e_{(1,j_1)}, c_{(1,j_1)})\} \ldots\ldots \{(e_{(r,1)}, c_{(r,1)}), (e_{(r,2)}, c_{(r,2)}) \ldots, (e_{(r,j_r)}, c_{(r,j_r)})\} \rangle$, where $e_{(\ell,i_\ell)}$ is the i-th event in the ℓ-th set of events and $c_{(\ell,i_\ell)}$ is the associated confidence value. Given a sequence $s = \langle s_1, \ldots, s_q \rangle$, we create $(q+1) \times (r+1)$ matrix $A_{i,s}[0..q][0..r]$ similar to an SLU (or a TLU) database; whilst the initialisations for computing the source support probability in an ELU database are similar to an SLU

(or a TLU) database, we compute the value c^*_{kl} in an ELU database as follows. For $1 \leq k \leq q$ and $1 \leq \ell \leq r$, let x_i be a variable whose value is $c_{(\ell,i_\ell)}$ if $s_k \subseteq e_{(\ell,i_\ell)}$ and 0 otherwise, then:

$$c^*_{kl} = \sum x_i. \tag{5.6}$$

We can then use Equation 5.5 similar to the SLU (or TLU) case to compute the source support probability and hence, the expected support of a sequence using Lemma 5.4.

Theorem 5.6. *Given a sequence s and an ELU database D^p, we can calculate the expected support of s in D^p and hence evaluate the interestingness predicate, in $O(kN)$ time.*

Proof. The proof is similar to that of Theorem 5.5 as we assume that the test in Equation 5.6 can be done in constant time in case of an ELU database as well.

□

5.1.3 SLU-D Case

We now discuss computing the expected support of a sequence in an SLU-D database. We first review a few concepts from Chapter 4. An SLU-D database D^p is a set of two relations, the activity relation $\mathcal{A}$ having records of the form (eid, e, sid) where eid is an event-id, e is an event and sid is the source-id (records are ordered by eid), and the mapping relation $\mathcal{M}$ having records of the form (sid, W) where every sid is associated with a distribution W over $\mathcal{S}$. Further, an sid-sequence is a sequence of all the events associated with an sid (ordered by a time-stamp); thus, an SLU-D database can be seen as a collection of sid-sequences where there is uncertainty about an sid-sequence being associated to a source σ. For an example, see the SLU-D database of Figure 4.4 transformed to sid-sequences in Table 4.10.

Consider a sample database D^p having two tuples in the activity relation $(e_1, (a), sid_0)$ and $(e_2, (a), sid_0)$ and one tuple in the mapping relation $(sid_0, (X : 0.7, Y : 0.3))$. The D^p in *sid*-sequence format is as follows: $(\langle (a)(a) \rangle, (X : 0.7, Y : 0.3))$. Now, consider computing the expected support of s in D^p in a way similar to the SLU case. As $D^p_X = \langle (a : 0.7)(a : 0.7) \rangle$ and $D^p_Y = \langle (a : 0.3)(a : 0.3) \rangle$, the contribution from source X and source Y to the expected support of s is 0.91 and 0.51 respectively, and thus the expected support of s in D^p is $0.91 + 0.51 = 1.42$; which is not right, as for this sample database, there are only two possible worlds: one where $\langle (a)(a) \rangle$ is associated with source X and the probability of this world is 0.7, and other when it is associated with source Y and the probability of this world is 0.3. Thus, the expected support of s in D^p is 1, not 1.42. We now show that the difficulty is not in taking one particular approach or with the linearity of expectation but is with the independence assumption needed to compute the source support probability and thus, is inherent to the problem. We define the following problem:

Definition 5.7 (Expected Support Problem)**.** Given an SLU-D database D^p, a sequence s, and an expected support threshold θ, is $ES(s, D^p) \geq \theta$, i.e. is the expected support of s in D^p is at least θ.

Theorem 5.8. Expected Support *is NP-complete.*

Proof. We reduce the 3D Matching problem, a known NP-complete problem [70] to the Expected Support problem. We first define the 3D Matching problem.

Definition 5.9 (3D Matching Problem)**.** Given three disjoint sets X, Y and Z, $|X| = |Y| = |Z| = N$, and a set of triples $T \subseteq X \times Y \times Z, |T| = M$, is there a set of triples $T' \subseteq T$, $|T'| = N$, such that no two triples in T' agree in any coordinate.

Given three disjoint sets X, Y and Z, each of size N, and a set of triples $T \subseteq X \times Y \times Z$, we create an instance of Expected Support (an SLU-D database D^p,

TABLE 5.2: An instance of 3D MATCHING.

Three disjoint sets:	$X = \{x_1, x_2\}, Y = \{y_1, y_2\}, Z = \{z_1, z_2\}$
Set of Triples:	$T = \{(x_1, y_1, z_2), (x_2, y_1, z_2), (x_2, y_2, z_1)\}$

a sequence s, and a number θ) in polynomial time such that solving the latter returns the answer to the former that is, given D^p, s and θ, we ask that is $ES(s, D^p) \geq \theta$? If the answer to this question is yes, we can also tell that there is a set of triples T' of size N such that no two triples in T' agree in any coordinate, and if the answer is no, then such a set of triples does not exist.

Let $X = \{x_1, \ldots, x_N\}$, $Y = \{y_1, \ldots, y_N\}$, and $Z = \{z_1, \ldots, z_N\}$ be the three disjoint sets, and T be the set of triples of size M, where a triple is of the form $(x_i, y_j, z_k) \in T$, $i, j, k \leq N$.

We create an instance of EXPECTED SUPPORT as follows. We generate an activity relation $\mathcal{A}$ of the form (eid, e, sid), by creating a multi-set T_X generated by selecting the x-coordinates from each triple (x_i, y_j, z_k) in T.

Note that $|T_X| = M$. For every $x_i \in T_X$ (taken as a multi-set), we take x_i as an event and time-stamp x_i from $(1, \ldots, M)$ in such a way that for any $x_i, x_{i'} \in T_X$, $i \neq i'$, if $i < i'$ then the time-stamp of all the events (x_i) is earlier than that of $(x_{i'})$. Similarly, we generate T_Y and T_Z and time-stamp the elements in T_Y and T_Z from $(M + 1, \ldots, 2M)$ and $(2M + 1, \ldots, 3M)$, respectively. We label every triple in T with an *sid* from $(1, \ldots, M)$ and thus, any of the x-, y- or z-coordinates in a triple are associated with the same *sid* in $\mathcal{A}$. Finally, we introduce an additional tuple $(3M + 1, (\$), 0)$ in $\mathcal{A}$, where $3N + 1$ is the time-stamp, ($) is the event, and 0 is the *sid*.

For example, consider the sample 3D MATCHING instance in Table 5.2 having three sets, $X = \{x_1, x_2\}$, $Y = \{y_1, y_2\}$ and $Z = \{z_1, z_2\}$, and a set of triples $T =$

Activity		
eid	*e*	*sid*
eid_1	(x_1)	sid_1
eid_2	(x_2)	sid_2
eid_3	(x_2)	sid_3
eid_4	(y_1)	sid_1
eid_5	(y_1)	sid_2
eid_6	(y_2)	sid_3
eid_7	(z_1)	sid_3
eid_8	(z_2)	sid_1
eid_9	(z_2)	sid_2
eid_{10}	(\$)	sid_0

Mapping	
sid	*W*
sid_0	$(\sigma_0 : 1.0)$
sid_1	$(\sigma_0 : \epsilon)(\sigma_1 : (1-\epsilon))$
sid_2	$(\sigma_0 : \epsilon)(\sigma_2 : (1-\epsilon))$
sid_3	$(\sigma_0 : \epsilon)(\sigma_3 : (1-\epsilon))$

FIGURE 5.1: The sample 3D MATCHING instance of Table 5.2 transformed to an SLU-D database D^p.

$\{(x_1, y_1, z_2),\ (x_2, y_1, z_2),\ (x_2, y_2, z_1)\}$. We transform our example 3D MATCHING instance to an SLU-D database D^p as follows. We first create the activity relation $\mathcal{A}$. We generate a multi-set $T_X = \{x_1, x_2, x_2\}$ and time-stamp the elements in T_X, taken as events in $\mathcal{A}$ as $(1, 2, 3)$. Similarly, we generate $T_Y = \{y_1, y_1, y_2\}$ and $T_Z = \{z_1, z_2, z_2\}$ and time-stamp the elements therein as $(4, 5, 6)$ and $(7, 8, 9)$, respectively. Next, we label every triple in T with an *sid* as $(1, 2, 3)$. As a final step in the creation of $\mathcal{A}$, we introduce an additional tuple $(10, \$, 0)$ in $\mathcal{A}$ where 10 is the time-stamp, (\$) is the event, and 0 is the *sid*.

TABLE 5.3: The sample SLU-D database of Figure 5.1 transformed to *sid*-sequences.

sid-sequences		W
sid_0:	$\langle(\$)\rangle$	$(\sigma_0 : 1.0)$
sid_1:	$\langle(x_1)\ (y_1)\ (z_2)\rangle$	$(\sigma_0 : \epsilon)(\sigma_1 : (1-\epsilon))$
sid_2:	$\langle(x_2)\ (y_1)\ (z_2)\rangle$	$(\sigma_0 : \epsilon)(\sigma_2 : (1-\epsilon))$
sid_3:	$\langle(x_2)\ (y_2)\ (z_1)\rangle$	$(\sigma_0 : \epsilon)(\sigma_3 : (1-\epsilon))$

Next, we generate the mapping relation $\mathcal{M}$ of the form (sid, W) by taking the distinct *sid*s in $\mathcal{A}$, and for every *sid* i, generate the distribution W by mapping *sid* i to source σ_i with probability $(1-\epsilon)$ and with source σ_0 with probability ϵ. Note that ϵ is a binary number (see Remark A.2). The value of ϵ will be chosen later, but it will be less than $1/2$. The *sid* 0 is associated only with source σ_0 with probability 1. Thus, we create an SLU-D database D^p having an activity and a mapping relation.

In the example 3D MATCHING instance, we generate the mapping relation by selecting *sid*s 1 and 2 from $\mathcal{A}$, and by associating those (i.e. *sid*s 1 and 2) with sources σ_1 and σ_2, respectively, with probability $(1-\epsilon)$, and with the source σ_0 with probability ϵ (ϵ will be chosen later). *sid* 0 is associated with source σ_0 with probability 1. An example of this transformation is shown in Figure 5.1. The sample SLU-D database of Figure 5.1 is transformed to *sid*-sequences in Table 5.3.

Next, we generate the sequence s by selecting the elements in X, Y and Z, and place them in order and then add \$ to the end to obtain $s = \langle x_1, \ldots, x_N, y_1, \ldots, y_N, z_1, \ldots, z_N, \$\rangle$. In the example, we generate $s = \langle x_1, x_2, y_1, y_2, z_1, z_2, \$\rangle$.

Finally, we will specify a θ depending on ϵ, M, and N. This essentially completes the creation of an instance of EXPECTED SUPPORT, which takes polynomial time.

We now show that we can choose ϵ and θ so that the answer to EXPECTED SUPPORT is yes, if and only if the answer to 3D MATCHING is yes. In this instance of

EXPECTED SUPPORT, a set of *sid*-sequences R is mapped to source σ_0 in every possible world. It is clear that a possible world supports the sequence s only if the source σ_0 supports it, due to the terminating \$ symbol. From the perspective of expected support computation, we only need to know if there exists a set R of size N that when mapped to source σ_0 in a possible world, supports s. Thus, we are only interested in such possible worlds where a set R of size N is mapped to source σ_0. We consider the following two cases:

(a) If the answer to 3D MATCHING is yes, there exists (at least) one set R of size N that when mapped to source σ_0 in a possible world supports s.

(b) If the answer to 3D MATCHING is no, then the possible world corresponding to the smallest set R that when mapped to source σ, supports s is at least of size $N + 1$.

Observe that a possible world where N out of M *sid*-sequences are mapped to source σ_0 has a probability $\epsilon^N * (1 - \epsilon)^{M-N}$. We choose the support threshold $\theta = \epsilon^N * (1 - \epsilon)^{M-N}$. We now show that if (a) holds then $ES(s, D^p) \geq \theta$, and if (b) holds then $ES(s, D^p) < \theta$.

We first consider case (a) which is obvious, as even if there is only one such possible world where s is supported, $ES(s, D^p) = \theta$. Clearly, if there are more than one such possible worlds, the expected support of s in D^p will be more than θ; thus, if (a) holds $ES(s, D^p) \geq \theta$.

We now focus on case (b). Suppose that R, $|R| = r \geq N + 1$, is the smallest set of *sid*-sequences that when mapped to source σ_0 in a possible world, then that possible world supports the sequence s. Clearly, there can be many R, and a possible world where any superset R' of R, $|R'| \leq M$, is mapped to source σ_0 will also contribute to $ES(s, D^p)$.

We want to choose ϵ is such a way that the contribution to the expected support of a sequence s from all such possible worlds where the *sid*-sequences in R or any of its superset R' are mapped to source σ_0, does not exceed θ. We show that for $\epsilon = (1/2^{M+1})$, the expected support of s in D^p cannot exceed θ. As $|R| = r \geq N+1$, $ES(s, D^p)$ will at most be:

$$\begin{aligned} ES(s, D^p) &\leq \binom{M}{r} \ (\epsilon^r * (1-\epsilon)^{M-r}) \\ &\quad + \\ &\quad \binom{M}{r+1} \ (\epsilon^{r+1} * (1-\epsilon)^{M-r-1}) \\ &\quad + \\ &\quad \vdots \\ &\quad + \\ &\quad \binom{M}{M} \quad (\epsilon^M * (1-\epsilon)^{M-M}) \\ ES(s, D^p) &\leq \left[\binom{M}{r} + \ldots + \binom{M}{M}\right] \ \epsilon^r * (1-\epsilon)^{M-r} \\ &\leq \quad 2^M \ * \ (\epsilon^r * (1-\epsilon)^{M-r}) \end{aligned}$$

we need to show that $2^M \ * \ \epsilon^r * (1-\epsilon)^{M-r} < \theta$ for $\epsilon = (1/2^{M+1})$, where $\theta = \epsilon^{r-1} * (1-\epsilon)^{M-r-1}$. We can write:

$$2^M \ * \ \epsilon^r * (1-\epsilon)^{M-r} < \epsilon^{r-1} * (1-\epsilon)^{M-r-1}$$

Solving above equation on both sides, we get:

$$2^M \ * \ \epsilon < (1-\epsilon),$$

which clearly holds for $\epsilon = (1/2^{M+1})$, and this proves the case (b).

Hence, we have shown that if we can answer the question, is $ES(s, D^p) \geq \theta$, we can also tell if there exists a set of triples T' of size N such that no two triples in T' agree in any coordinate, and thus reducing the 3D MATCHING problem to the EXPECTED SUPPORT problem, and showing that EXPECTED SUPPORT is NP-complete.

□

We now give examples to explain each of case (a) and (b). We first discuss a situation where (a) holds. Consider the sample example in Table 5.2, we generate $s = \langle x_1, x_2, y_1, y_2, z_1, z_2, \$\rangle$, and set the value of $\epsilon = (1/2^{3+1})$, as $M = 3$. Next, we choose $\theta = (1/2^{3+1})^2 * (1 - 1/2^{3+1})^{3-2}$, as $|N| = 2$.

If (a) holds that is if the answer to this instance of 3D MATCHING is yes, there must at least be one set R containing exactly 2 *sid*-sequences that when mapped to source σ_0 in a possible world supports s. In our example, there exists such a set R containing exactly two *sid*-sequences, i.e. *sid*-sequence 1 and 3 that when mapped to source σ_0 support s. A superset R' of R containing all three *sid*-sequences will also contribute to $ES(s, D^p)$. Thus, $ES(s, D^p) = ((1/2^4)^2 * (1 - (1/2^4))^{3-2}) + ((1/2^4)^3 * (1 - (1/2^4))^{3-3})$, which clearly is greater than $\theta = (1/2^4)^2 * (1 - (1/2^4))^{3-2}$.

We now discuss an example where (b) holds. In contrast with the previous example, consider the set of triples $T = \{(x_1, y_1, z_2), (x_2, y_1, z_1), (x_1, y_2, z_1)\}$. We then generate the sequence $s = \langle x_1, x_2, y_1, y_2, z_1, z_2, \$\rangle$, and set the value of $\epsilon = (1/2^{3+1})$, and choose $\theta = (1/2^{3+1})^2 * (1 - 1/2^{3+1})^{3-2}$, as before.

If (b) holds that is if the answer to this instance of 3D MATCHING is no, the smallest set R containing a set of *sid*-sequences that when mapped to source σ_0 in a possible world supports s, must at least be of size 3. It can be seen that in our example, the

smallest such set of *sid*-sequences is of size 3, i.e. containing *sid*-sequences $1, 2$ and 3. Thus, $ES(s, D^p) = (1/2^4)^3 * (1 - (1/2^4))^{3-3}$, which clearly is less than θ.

5.2 Probabilistic Frequentness Computation

We now discuss Problem 5.2, that is, we focus on evaluating the interestingness predicate under the probabilistic frequentness measure for the uncertainty models in Chapter 4. As already mentioned, that due to potential exponential number of possible worlds, it is infeasible to evaluate the interestingness predicate directly by using Equation 4.2. We show that whilst we can evaluate the interestingness predicate efficiently for the TLU and ELU model, it is computationally intractable to evaluate interestingness predicate for the SLU and SLU-D model.

We first show that for the TLU and ELU model, we can compute the probabilistic frequentness of a sequence in a probabilistic database D^p efficiently using dynamic programming.

5.2.1 TLU/ELU Case

Given a sequence s and a TLU (or an ELU) database D^p, we compute the probabilistic frequentness of s in D^p as follows. We compute the entire *support probability distribution*. The support probability distribution is the vector $\langle \Pr_0(s), \ldots, \Pr_m(s) \rangle$, where $\Pr_k(s)$ is the probability that the support of s is exactly k. Given the support probability distribution, we can then calculate the probability that support of s is at least θ, denoted by $\Pr_{\geq\theta}$, as follows:

$$\Pr_{\geq\theta}(s) = \sum_{k=\theta}^{m} \Pr_k(s), \tag{5.7}$$

in $O(m)$ time and thereby answer the interestingness predicate.

As already mentioned that in a TLU (or an ELU) database, the p-sequences are independent. This allows the support probability distribution to be computed as a dynamic programming recurrence as follows. We first compute $\Pr(s \preceq D_i^p)$ for all sources i in $O(kN)$ time as in Theorem 5.5. Next, we define $\Pr_{i,j}(s)$, for $0 \leq i, j \leq m$, as the probability that exactly i of the first j sources support s. We then use the formula:

$$\Pr_{i,j}(s) = \Pr_{i-1,j-1}(s) \cdot \Pr(s \preceq D_i^p) + \Pr_{i,j-1}(s) \cdot (1 - \Pr(s \preceq D_i^p)), \tag{5.8}$$

where $\Pr_{0,j}(s) = 1,\ 0 \leq j \leq m$ and $\Pr_{i,j}(s) = 0$, for all $i > j$, to compute all the values $\Pr_{i,j}$ in $O(m^2)$ time. Since $\Pr_{i,m}(s) = \Pr_i(s)$, we get the full support probability distribution and can use this to determine if s is a probabilistic frequent sequence; the overall time is $O(kN + m^2)$. In summary:

Theorem 5.10. *Given a sequence s and a TLU or an ELU database D^p, we can calculate the support probability distribution, and hence evaluate the interestingness predicate in $O(kN + m^2)$ time.*

Remark 5.11. When computing $\Pr_{i,j}(s)$ using Equation 5.8, we consider two cases: either σ_i supports s, and exactly $j-1$ of $\sigma_1, \ldots, \sigma_{i-1}$ support s (the first term in Equation 5.8) or σ_i does not support s and exactly j of $\sigma_1, \ldots, \sigma_{i-1}$ support s (the second term in Equation 5.8). The correctness of Equation 5.8 depends crucially on the fact that we assume independence among p-sequences, so $\Pr(s \preceq D_i^p)$ (respectively $\Pr(s \not\preceq D_i^p)$) is unchanged even when conditioned on knowing that exactly $j-1$ (respectively j) of the first $i-1$ sources support s.

Remark 5.12. Since N/m is the average length of a p-sequence, $N/m \ll m$ and (since k is not very large as well) the m^2 term will often dominate the kN term.

This means the interestingness predicate for computing probabilistic frequentness will often be computationally more expensive than for computing expected support.

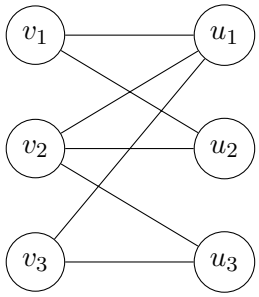
5.2.2 SLU Case

In an SLU database, we cannot use Equation 5.8 to efficiently compute the support probability distribution, since the p-sequences are not independent (see Remark 5.11). Consider the very simple probabilistic database which consists of just the event $\{a\}$, associated with source σ_1 and σ_2 with probabilities 0.5 each. There are only two possible worlds, the first with the event $\{a\}$ associated with σ_1 and nothing with σ_2, and the second the other way around. Clearly, if s is the sequence $\langle\{a\}\rangle$, then $\Pr(s \preceq D_1^p) = \Pr(s \preceq D_2^p) = 0.5$. However, applying Equation 5.8 gives that $\Pr_{1,1} = 0.5$ (correct) and $\Pr_{2,2} = 0.25$ (incorrect). The probability that both sources support s is zero, not 0.25 – there is no possible world in which both sources support s. To see what goes wrong, consider the computation of $\Pr_{2,2}(s)$ as $\Pr_{1,1}(s) \cdot 0.5 + \Pr_{2,1}(s) \cdot 0.5 = 0.5 \cdot 0.5 + 0 \cdot 0.5 = 0.25$. Looking at the first term, if σ_1 supports s, then conditioned on this knowledge, the probability that σ_2 supports s is zero. Thus, the correct computation for $\Pr_{2,2}(s)$ would be as $0.5 \cdot 0 + 0 \cdot 0.5 = 0$. Unfortunately, the difficulty is not with one particular approach but is intrinsic to the problem: we now show that computing even a single entry of the support probability distribution is provably hard. Define the following problem:

Definition 5.13 (Exact-k Support Problem)**.** The input is an SLU database D^p, a sequence s and a number k, $0 \leq k \leq m$. The output is $\Pr_k(s)$, i.e. the probability that exactly k sources support s.

Theorem 5.14. Exact-k Support *is #P-complete.*

Proof. We reduce the problem of computing the number of perfect matchings in a bipartite graph, a known #P-complete problem [71] to Exact-k Support.



(a)

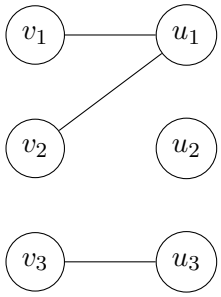
eid	*e*	*W*
e_1	(a)	$(\sigma_1{:}0.5)(\sigma_2{:}0.5)$
e_2	(a)	$(\sigma_1{:}0.33)(\sigma_2{:}0.33)(\sigma_3{:}0.33)$
e_3	(a)	$(\sigma_1{:}0.5)(\sigma_3{:}0.5)$

(b)

FIGURE 5.2: A sample bipartite graph (a) transformed to a probabilistic database (b). The vertices in V correspond to *eid* and the vertices in U correspond to source $\sigma_i \in \mathcal{S}$.

source	event
σ_1	e_1, e_2
σ_2	$\langle\rangle$
σ_3	e_3

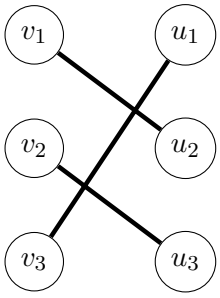
$\Longrightarrow$



(a)

source	event
σ_1	e_3
σ_2	e_1
σ_3	e_2

$\Longrightarrow$



(b)

FIGURE 5.3: Two possible worlds in $PW(D^p)$ and their bipartite graph representations. A perfect matching (b), when every vertex in $U \cup V$ is adjacent to a single edge.

Let $G(U, V, E)$ be an undirected bipartite graph, where U and V are disjoint sets of *vertices* and E is the set of *edges* between them, $E \subseteq U \times V$. We assume that $|U| = |V| = n$. A *perfect matching* M is a subset of edges in E such that each vertex in $U \cup V$ is adjacent to exactly a single edge in M. Given a bipartite graph G, the problem of counting how many perfect matchings there are in G is #P-complete [71].

Given a bipartite graph G, to compute the number of matchings in G, we create an instance of Exact-k Support (a probabilistic database D^p, a sequence s and a number k) in polynomial time such that solving the latter instance gives the number of perfect matchings in G. Given $G = (U, V, E)$ where $U = \{u_1, \ldots, u_n\}$ and $V = \{v_1, \ldots, v_n\}$, we create a set of sources $\mathcal{S} = \{\sigma_1, \ldots, \sigma_n\}$ such that $\sigma_i \in \mathcal{S}$ represents $u_i \in U$. The probabilistic database D^p is a set of records $r_i = (e_i, e, W_i)$, where e_i is a event id, e is an event and W_i is a distribution over sources. The record r_i represents v_i in V together with all the edges from v_i to the *neighbourhood* of v_i, i.e. the set of vertices in U adjacent to v_i. In what follows, we denote the neighbourhood of v_i as $N(v_i)$. The event contained in *every* record is a set containing just the singleton element $\{a\}$. In the i-th record r_i, the distribution W_i contains only the sources σ_j that represent vertices $u_j \in N(v_i)$. All sources in W_i have the same probability, i.e. $(1/|N(v_i)|)$. An example of such a transformation is shown in Figure 5.2. Finally, we choose $k = 0$ and the sequence $s = (a)(a)$, and ask to compute $\Pr_0(s)$, i.e. the probability that no sources support s. This completes creation of the instance of Exact-k Support, and the transformation is clearly polynomial-time.

Clearly, every possible world $D^* \in PW(D^p)$ is equally likely, and the probability of a possible world D^* is $\phi = (1/|PW(D^p)|)$, where $|PW(D^p)| = \prod_{i=1}^{n} |N(v_i)|$. For example, there are 12 possible worlds for the sample SLU database in Figure 5.2(b), and the probability of each world is $(1/12)$. In each possible world, each record r_i is associated with some source σ_j, so a possible world can be viewed as a sub-graph

of G where every vertex in V has degree exactly one. Two possible worlds and their corresponding graphs are shown in Figure 5.3. Those possible worlds where each source is associated with exactly one record corresponds to a perfect matching (Figure 5.3(b)); in such possible worlds, the support of the sequence $s = (a)(a)$ will clearly be zero. Thus, we see that $\Pr_0(s) = \phi \cdot (\#$ matchings in $G)$, and once we are given $\Pr_0(s)$, we obtain the number of matchings in G by multiplying by the total number of possible worlds. For example, in database in Figure 5.2(b), there are only three possible worlds where each source is associated with exactly one event, which are $(\sigma_1 : e_1,\ \sigma_2 : e_2,\ \sigma_3 : e_3)$, $(\sigma_1 : e_2,\ \sigma_2 : e_1,\ \sigma_3 : e_3)$ and $(\sigma_1 : e_3,\ \sigma_2 : e_1,\ \sigma_3 : e_2)$. Hence, $Sup(s, D^*) = 0)$ in three worlds and therefore, $Pr_0(s) = 3 \times (1/12) = 0.25$. We multiply the answer by the number of possible worlds, 12, to get 3, the number of perfect matchings in G.

Hence, we have shown that if the value $\Pr_k(s)$ can be computed for $s = (a)(a)$ and $k = 0$ in D^p, we can also find number of perfect matchings in G, thus reducing the problem of counting perfect matchings in a bipartite graph to Exact-k Support, and showing that Exact-k Support is #P-complete.

□

5.2.3 SLU-D Case

Remark 5.15. We have shown in Theorem 5.8 that it is NP-complete to compute the expected support of a sequence s in an SLU-D database. As computing the probabilistic frequentness of a sequence s in an SLU-D database is computationally no less hard than computing the expected support of a sequence s (probabilistic frequentness is the confidence in the support of a sequence), we conjecture that a polynomial time algorithm is unlikely to exist for computing the probabilistic frequentness of a sequence in an SLU-D database.

5.3 Summary

We have formulated two problems based on the interesting measures proposed in Chapter 4, namely expected support and probabilistic frequentness, for evaluating the interestingness predicate for the uncertainty models we consider, namely TLU, ELU, SLU and SLU-D. We have shown that different formulations of the probabilistic SPM problem lead to very different outcomes from a complexity theoretic viewpoint. For example, we have shown that whilst we can compute the probabilistic frequentness of a sequence in a TLU database in polynomial time, it is #P-complete to do so for the SLU case.

Chapter 6

Probabilistic SPM Algorithms

A classical SPM algorithm enumerates all frequent sequences by exploring the search space and by applying the interestingness predicate. We have shown in Chapter 5 that it may or may not be possible to evaluate the interestingness predicates that we defined in Chapter 4 in polynomial time. For example, we have shown that for the SLU model, it is #-P complete to compute the probabilistic frequentness of a sequence, although we can compute the expected support of a sequence in polynomial time. We focus on the latter in this chapter, that is, we compute expected frequent sequences (see Definition 4.4) in an SLU database. In Chapter 5, we have also shown that it is possible to evaluate the interestingness predicate for expected support in the TLU and ELU models in polynomial time but we do not take this any further.

As discussed in Chapter 2, the search space exploration in classical SPM algorithms is generally based either on the candidate generation or the pattern growth framework. In this chapter, we propose probabilistic SPM algorithms based on each of these frameworks.

We first describe probabilistic SPM algorithms based on the candidate generation framework.

6.1 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 exploration (DFS) of the search space. Our proposed BFS approach is similar to GSP [38], and our DFS approach is similar to the depth-first variant of SPADE [36] (SPADE has a breadth-first variant as well) or SPAM [34], in search space exploration.

Although the dynamic programming (DP) algorithm proposed in Section 5.1 for computing the source support probability and hence, evaluating the interestingness predicate is faster than evaluating the interestingness predicate directly by using Equation 4.1, there could potentially be a large number of candidate sequences which need to be tested for being frequent, and thus the DP computation needs to be performed over and over again. We first give some optimizations to speed up the support computation task.

6.1.1 Optimization

In this section, we describe two optimized sub-routines:

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.

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

(i) *Apriori Pruning:* that is, pruning a candidate sequence if any of its subsequences is not frequent.

(ii) *Probabilistic Pruning:* that is, eliminating potential infrequent candidate sequences without support computation.

We now elaborate on each of these below.

6.1.1.1 Fast Frequent 1-sequence Computation

The DP algorithm (Section 5.1) can be used to find all frequent 1-sequences by applying Lemma 5.4 and Equation 5.3. However, a naive approach, for example for each 1-sequence $\langle(x)\rangle$, $x \in \mathcal{I}$, computing $ES(\langle(x)\rangle, D^p)$ by applying Lemma 5.4 and Equation 5.3, will require $|\mathcal{I}|$ scans of the database. In classical SPM, the task of finding all frequent 1-sequences could be performed in linear time in a single scan over the database. We show that we can achieve the same, that is, we can find all frequent 1-sequences in a single scan over the database in linear time, in the probabilistic case as well. We first give a simple intuitive closed-form expression for computing the source support probability for 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:

$$\Pr[s \preceq D_i^p] = 1 - \prod_{\ell=1}^{r}(1 - c_{1\ell}^*). \tag{6.1}$$

The term on the right in Equation 6.1 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. We now verify by induction that using Equation 6.1 gives the same answer as we get by applying Lemma 5.4. As from Section 5.1, $A[0, \ell] = 1$ for all ℓ,

$0 \leq \ell \leq r$, Equation 5.5 can be written as:

$$A[1, \ell] = (1 - c^*_{1\ell}) * A[1, \ell - 1] + c^*_{1\ell}. \tag{6.2}$$

We first show that the induction hypothesis holds for $\ell = 1$, by substituting $\ell = 1$ in Equation 6.1 and in Equation 6.2. First, by putting $\ell = 1$ in Equation 6.2, we get $A[1, 1] = (1 - c^*_{11}) * A[1, 0] + c^*_{11} = c^*_{11}$, as $A[1, 0] = 0$. We get the same answer from Equation 6.1 for $\ell = 1$, $A[1, 1] = 1 - \prod_{\ell=1}^{1}(1 - c^*_{1\ell}) = 1 - (1 - c^*_{11}) = c^*_{11}$. Now, assume that the induction hypothesis holds for $\ell = t - 1$, then:

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

which proves Equation 6.1.

We now describe a procedure for computing the expected support of all 1-sequences s. Initialize two arrays F and G, each of size $|\mathcal{I}|$, to zero and consider each source i in turn. If $D^p_i = \langle (e_1, c_1), \ldots, (e_r, c_r) \rangle$, for $k = 1, \ldots, r$ take the pair (e_k, c_k) and iterate through each $x \in e_k$, setting $F[x] := (1 - c_k)$ if $F[x] = 0$, and setting $F[x] := F[x] * (1 - c_k)$ otherwise. Once we are finished with source i, we set the entries that have been updated in F to $F[x] := 1 - F[x]$ and then update $G[x] := G[x] + F[x]$ and reset $F[x]$ to zero (we use a linked list to keep track of which entries of F are updated for a given source). At the end, for any 1-sequence $s = \langle (x) \rangle$, where $x \in \mathcal{I}$, $G[x] = ES(s, D^p)$.

6.1.1.2 Incremental Support Computation

We show that considering a source i, for any two sequences s and t where t is an extension of s, we can reuse the DP rows computed for s to quickly compute $\Pr[t \preceq D_i^p]$. Recall from Section 2.3 that for a sequence s and an item $\{x\}$, a sequence t obtained by extending s with $\{x\}$ is an S-extension of s if $\{x\}$ is added to the end of s as a new element, and if t is obtained by adding $\{x\}$ to s as part of the last element in s, t is an I-extension of s. 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. While computing $\Pr[t \preceq D_i^p]$, we will exploit the similarity between s and t to compute $\Pr[t \preceq D_i^p]$ more rapidly. Say that t is an S-extension of $s = \langle s_1, \ldots, s_q \rangle$, then the first q DP rows for s and t are exactly the same, and we can start the DP computation for t from the $(q+1)$-st row assuming that the q-th row is available. Similarly, for an I-extension t of s, if the first k, $k < q$, elements in s and t are exactly the same, we can start the DP computation for t from the $(k+1)$-st element assuming that the DP row for the k-th element is available. For example, if $s = \langle (a)(b,c) \rangle$ and $t = \langle (a)(b,c)(d) \rangle$ (t is 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 and therefore, could be reused in support computation. Similarly, for an I-extension $t = \langle (a)(b,c,d) \rangle$ of $s = \langle (a)(b,c) \rangle$, we could reuse the DP row for $\langle (a) \rangle$.

We now describe how we can reuse previously computed rows of the DP matrix 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 $\Pr[t \preceq D_i^p] = B_{i,t}[r]$:

Algorithm 1 Incremental Support Computation (I-extension case)

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

Lemma 6.1. *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. Equation 5.5).

Now consider the case where t is a I-extension, i.e. $t = \langle s_1, \ldots, s_q \cup \{x\} \rangle$ for some $x \notin s_q$. Firstly, observe that since the first $q-1$ elements of s and t are pairwise equal, the first $q-1$ rows of $A_{i,s}$ and $A_{i,t}$ are also equal. The $(q-1)$-st row of $A_{i,s}$ is enough to compute the q-th row of $A_{i,t}$, but we only have $B_{i,s}$, the $(q-1)$-st row of $A_{i,s}$. In general we cannot calculate the entire $(q-1)$-st row of $A_{i,s}$ from the q-th row, 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 to the next value of ℓ. If $t_q \subseteq e_\ell$, then $s_q \subseteq e_\ell$ and so:

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

TABLE 6.1: 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 $\Pr[t \preceq D^p_X]$ in the SLU database of Figure 4.3. Note that the row corresponding to $\langle (a) \rangle$ is *not* available.

		$(a,d,e:0.6)$	$(b,c:0.3)$	$(a,b,c:0.7)$	$(c,d,e:1.0)$
	A[0,0] = 1	A[0,1] = 1	A[0,2] = 1	A[0,3] = 1	A[0,4] = 1
$\langle (a) \rangle$	A[1,0] = 0	**A[1,1]** = 0.4× A[1,0] + 0.6× A[0,0] = 0.6	**A[1,2]** = 0.6	A[1,3] = 0.3× A[1,2] + 0.7× A[0,2] = 0.72	A[1,4] = 0.72
$\langle (a)(b) \rangle$	A[2,0] = 0	A[2,1] = 0	A[2,2] = 0.7× A[2,1] + 0.3× **A[1,1]** = 0.18	A[2,3] = 0.3× A[2,2] + 0.7× **A[1,2]** = 0.474	A[2,4] = 0.474
$\langle (a)(b,c) \rangle$	A[3,0] = 0	A[3,1] = 0	A[3,2] = 0.7× A[3,1] + 0.3× **A[1,1]** = 0.18	A[3,3] = 0.3× A[3,2] + 0.7× **A[1,2]** = 0.474	A[3,4] = 0.474

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 6.1.

6.1.1.3 Apriori Pruning

We show that an apriori property (recall Definition 2.7) holds in the probabilistic settings as well.

Lemma 6.2 (Apriori Property). *Given two sequences s and t, and a probabilistic database D^p, if s is a subsequence of t, then $ES(s,D^p) \geq ES(t,D^p)$.*

Proof. 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 Equation 4.1 we have,

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

□

6.1.1.4 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 6.3. *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 Equation 5.4. We now prove by induction on ℓ that:

$$A[q, \ell] \leq A[q-1, \ell] * p_\ell, \tag{6.3}$$

substituting $\ell = r$ in Equation 6.3 proves the lemma. We now prove Equation 6.3, which clearly holds for $\ell = 0$ since $A[q, 0] = A[q-1, 0] = p_0 = 0$. Subsequently:

$$\begin{aligned}
A[q,\ell] &= (1 - c^*_{q\ell}) * A[q, \ell-1] + c^*_{q\ell} * A[q-1, \ell-1] \quad \text{(Equation 5.5)} \\
&\leq (1 - c^*_{q\ell}) * A[q-1, \ell-1] * p_{\ell-1} + c^*_{q\ell} * A[q-1, \ell-1] \quad \text{(Equation 6.3)} \\
&\leq A[q-1, \ell-1] * \left((1 - c^*_{q\ell}) * p_{\ell-1} + c^*_{q\ell}\right) \\
&\leq A[q-1, \ell-1] * p_\ell \\
&\leq A[q-1, \ell] * p_\ell.
\end{aligned}$$

□

We now indicate how Lemma 6.3 is used. Suppose, for example, that we have a candidate sequence $s = \langle (a)(b,c)(a) \rangle$, and a source X. By Lemma 6.3:

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

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

6.1.2 Breadth-First Exploration

We give an overview of our BFS approach. We first review a few concepts from Chapter 2. The algorithm performs a series of steps repeatedly until the termination conditions for the algorithm are satisfied, and we call each execution of this series

of steps a *phase*. The length of a sequence is the number of items in it. A sequence having length j is called a j-sequence. The set of candidate j-sequences is called C_j and the set of frequent j-sequences is called L_j. For $j = 2$ onwards, the input to the j-th phase is the set L_{j-1}, and the output is the set L_j.

In our BFS approach, the algorithm first makes a pass over the SLU database D^p and computes all the frequent 1-sequences using the fast frequent 1-sequence computation (Section 6.1.1.1). Then, the following steps are performed in any phase $j \geq 2$. The set L_{j-1} is used to obtain C_j, and while generating C_j, apriori pruning is applied to C_j and thus, C_j contains only those candidate j-sequences that pass the apriori pruning. In addition to apriori pruning, probabilistic pruning can also be applied to C_j. Then, we perform the support computation for C_j, and the candidate j-sequences having support at least θ is the set of frequent j-sequences L_j. The algorithm stops when no more frequent sequences can be found, or when no more candidate sequences can be generated, and outputs all the frequent sequences. An outline of our BFS approach is in Algorithm 2.

Algorithm 2 Breadth-First Exploration

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

We now elaborate on the following key concepts:

- *Candidate Generation:* In the j-th phase, we generate the set of candidate j-sequences C_j from the set of frequent $(j-1)$-sequences L_{j-1}. The objective of candidate generation is to generate the smallest possible superset C_j of the set of frequent j-sequences L_j.

- *Support Computation:* We perform support computation for C_j as follows. We consider each source i in turn, and split the support computation task into the following two sub-tasks.

  - *Narrowing:* We only need to consider those candidate j-sequences that could potentially be supported by source i. Thus, the objective in narrowing is to find a smallest possible subset $N_{i,j}$ of the set of candidate j-sequences C_j such that no sequence in $C_j - N_{i,j}$ can be supported by source i.

  - *Computing Expected Support:* We compute the expected support for all the sequences $s \in N_{i,j}$, and while performing this task, we intend to reuse already computed results in order to save CPU cost.

6.1.2.1 Candidate Generation

We now elaborate on the candidate generation details for BFS. We first describe the procedure for generating candidate 2-sequences, and then discuss generating the candidate sequences in the subsequent phases.

In phase 2, we generate the set of candidate 2-sequences C_2 from the set of frequent 1-sequences L_1 as follows. For any two frequent 1-sequences $s, s' \in L_1$, we add s' to s as a separate element, i.e. we generate an S-extension of s, and we add s' to s as part of an existing element as well, if s' is lexicographically greater than s, i.e. we generate an I-extensions of s. For example, given two frequent 1-sequences $\langle(a)\rangle$

and $\langle(b)\rangle$, the possible candidate 2-sequences are $\langle(a)(a)\rangle$, $\langle(a)(b)\rangle$, $\langle(b)(a)\rangle$, $\langle(b)(b)\rangle$, and $\langle(a,b)\rangle$.

In the j-th phase, for $j = 3$ onwards, the set of frequent $(j-1)$-sequences L_{j-1} is joined with itself to generate the set of candidate j-sequences C_j. As in [38], we join two $(j-1)$-sequences s and s' if and only if the sequences that remain after deleting the first item in s and the last item in s' are the same. A j-sequence t obtained by joining two $(j-1)$-sequences s and s' is the sequence s extended with the last item $\{x\}$ in s'. The item $\{x\}$ is added to s in the same way as it was added to s', that is, $\{x\}$ is added as a separate element at the end of s if it was a separate element in s' (t is an S-extension of s), and is added to the last element of s if it was part of the last element in s' (t is an I-extension of s). This is illustrated by the following example, $s = \langle(a)(b,c)(d)\rangle$ joins with a sequence $\langle(b,c)(d)(e)\rangle$ to yield a sequence $\langle(a)(b,c)(d)(e)\rangle$. Similarly, s can be joined with a sequence $\langle(b,c)(d,e)\rangle$ to yield a sequence $\langle(a)(b,c)(d,e)\rangle$. However, s cannot be joined with $\langle(b)(c)(d)(e)\rangle$ as the remaining sequences after deleting the first item from s and the last item from $\langle(b)(c)(d)(e)\rangle$ – $\langle(b,c)(d)\rangle$ and $\langle(b)(c)(d)\rangle$ – are not the same. Similarly, s can not be joined with $\langle(b,c,d)(e)\rangle$ either, as $\langle(b,c)(d)\rangle$ does not match with $\langle(b,c,d)\rangle$.

Recall that the objective in candidate generation is to generate the smallest possible superset C_j of the set of frequent j-sequences L_j. To reduce the size of C_j, we apply pruning techniques to candidate j-sequence in C_j as follows. We first apply apriori pruning and consider every j-sequences $s \in C_j$ in turn, and if any of the $(j-1)$-subsequences of s is not in L_{j-1} (is not frequent), we delete s from C_j (by Lemma 6.2, s cannot be frequent). In addition, we can also apply probabilistic pruning for which we compute an upper bound on the expected support of s in D^p, and delete s from C_j if the upper bound value computed is less than the support threshold θ (Lemma 6.3). Note that apriori pruning has no effect on candidate 2-sequences and probabilistic pruning is the only possibility.

6.1.2.2 Support Computation

We now discuss the support computation details for BFS. As already mentioned, we split the support computation task into two sub-tasks, that is, narrowing and computing expected support. We elaborate on each of these below.

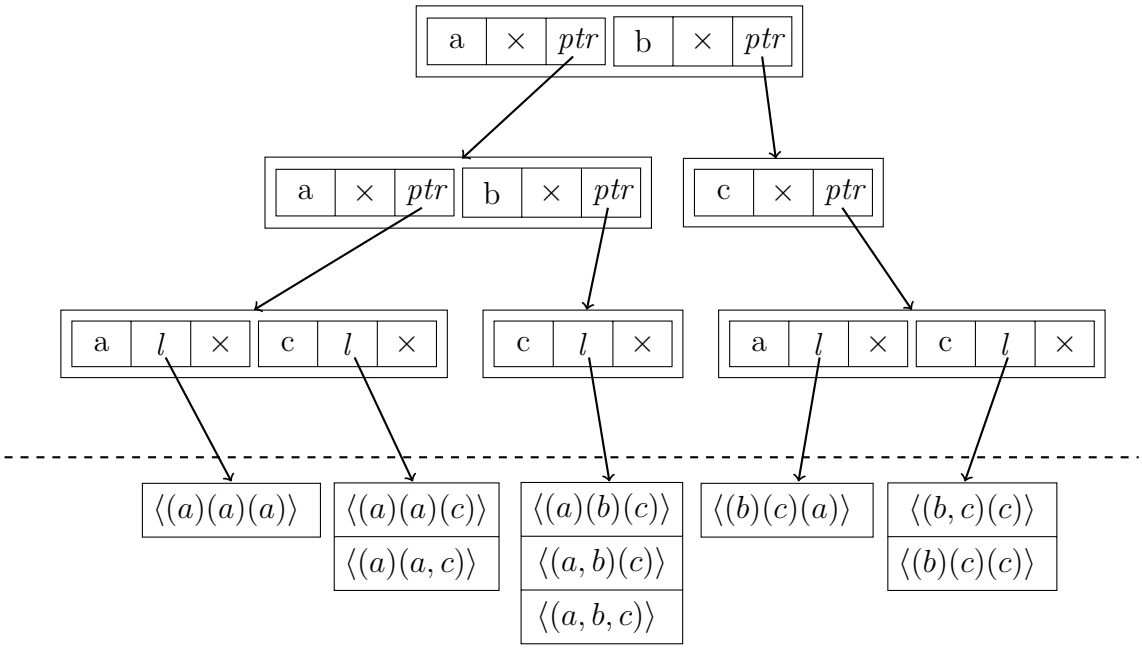
Narrowing

We define narrowing as follows. Given the set of candidate j-sequences C_j and a source i, the task of narrowing is to find a subset $N_{i,j}$ of the set of candidate j-sequences C_j that may be supported by source i. However, while $N_{i,j}$ should be of smallest possible size, $N_{i,j}$ must contain all the candidate j-sequences that have non-zero probability of being supported by source i. We denote the set of frequent j-sequences that have a non-zero probability of being supported by source i by $L_{i,j}$. After computing L_1, we store $L_{i,1}$ with each source for the entire duration of the algorithm. Further, for each $s \in L_{i,1}$, we also store $\Pr[s \preceq D_i^p]$, i.e. the probability with which source i supports s, when probabilistic pruning is used. We consider two ways of narrowing:

Prefix-based Narrowing

We store all the candidate sequences in C_j in a hash-table. An entry in the hash-table is a (key, value) pair of the form (s', ptr), where s' is a $(j-1)$-prefix (see Definition 2.10) of a sequence s and ptr is a pointer to a list of all the S- or I-extensions of s' in C_j. We populate the hash-table as follows. For every sequence $s \in C_j$, we split s into two parts, s' and $\{x\}$, where s' is a $(j-1)$-prefix of s and $\{x\}$ is the j-th item in s. We first check to see if the key s' is already in the hash-table and if yes, we add $\{x\}$ to the list of (S- or I-) extensions of s' pointed to by ptr. Otherwise,

FIGURE 6.1: A sample hashtree where each node is a set of triples of the form (k, ℓ, ptr). '×' represents a null pointer. The pointer ℓ at the leaf node points to a list of candidate sequences having the same characteristic string.



we add (s', ptr) to the hash-table as a (key, value) pair, where ptr is a pointer to a list containing the item $\{x\}$. Further, when recording an extension of s', we also keep track of the type of the extension (i.e. S- or I-extension). At the end of the $(j-1)$-st phase, we compute the sets $L_{i,j-1}$ for all sources i and keep them until the end of the j-th phase. When considering source i, for every sequence $s' \in L_{i,j-1}$, we use the hash-table to find all sequences $s \in C_j$ that are extensions of s'. Then, for every sequence $s \in C_j$ that is an S- or I-extension of s', we further check to see if $x \in L_{i,1}$ and if so, we add s to $N_{i,j}$.

Hashtree-based Narrowing

We store all the sequences in C_j in a *hashtree* in a manner that is slightly different from [38]. We first discuss how we construct the hashtree and then elaborate on how

it is used for narrowing. A *characteristic string* of a sequence is a string that contains all the items in the sequence, in the same order as they appear in the sequence. For example, the characteristic string of a sequence $\langle (a,b)(a,c) \rangle$ is *abac*. A node in a hashtree contains a set of triples of the form (k, ℓ, *ptr*), where k is an item, and ℓ and *ptr* are pointers. If a node in a hashtree is a non-leaf node, ℓ is null, and *ptr* contains a pointer to a node further down the hashtree; we say the label of *ptr* is k. If a node is a leaf node, *ptr* is null, and ℓ contains a pointer to a list of sequences which have the same characteristic string x, where x is a string that is obtained by concatenating all the labels in *ptr* starting from the root node down to the leaf node along with the leaf node item k. We generate the hashtree using the characteristic strings of the sequences in C_j. An sample hashtree is shown in Figure 6.1.

It is obvious that source i can potentially support only those candidate sequences in C_j whose characteristic strings have items only from $L_{i,1}$. When considering source i, we extract all the sequences in C_j that could be supported by source i as follows. For every item $y \in L_{i,1}$, we apply the hash function recursively on every item $z \in L_{i,1}$ starting from the root node until we get to a leaf node, and the leaf node contains a pointer ℓ to the list of sequences $s \in C_j$ which have the same characteristic string. We add the list of sequences pointed to by ℓ to $N_{i,j}$, and visit all possible leaf nodes that could be reached by applying hash function on every item in $L_{i,1}$ recursively thus, obtaining $N_{i,j}$.

Computing Expected Support

Given $N_{i,j}$ 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 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, then if s and t have a common prefix of length k, then we start the

computation of $\Pr[t \preceq D_i^p]$ from the $(k+1)$-st row in the DP matrix. Note that in the prefix-based narrowing, we sort the candidate sequences in a lexicographic order and thus the sequences with common prefixes are processed together. Similarly, in hashtree based narrowing, all the k-sequences in a leaf node have a common $(k-1)$ prefix and thus, only the k-the row in the DP matrix need to be computed for the sequences in a leaf node.

6.1.3 Depth-First Exploration

We now give an overview of our DFS approach. The algorithm first makes a pass over the SLU database D^p and computes all the frequent 1-sequences L_1. Assume that the sequences in L_1 are in ascending order. Next, starting with each frequent 1-sequence $s \in L_1$, s is both S- and I-extended with every valid item $x \in L_1$ and then for all extensions t of s, we first apply partial apriori pruning (Section 6.1.3.1) to t. In addition to partial apriori pruning, probabilistic pruning could also be applied. If t passes the pruning test, we compute the expected support of t in D^p and if t is found to be frequent, we keep exploring the search space by extending t using valid $x \in L_1$, recursively. The algorithm stops when no more candidate sequences can been generated or when no more frequent sequences can be found, and outputs all the frequent sequences. An outline of our DFS approach is in Algorithm 3 and Algorithm 4.

We now elaborate on two key details. We first discuss the candidate generation and then describe the support computation for DFS.

Algorithm 3 Depth-First Exploration

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

Algorithm 4 Depth-First Traversal

Input: SLU probabilistic database D^p and a sequence s.
Output: All possible extensions of s with $ES(s, D^p) \geq \theta$.
function TraverseDFS(s, L_1)
$L \leftarrow \emptyset$
for all valid $x \in L_1$ in order **do**
  $t \leftarrow \langle s \cdot \{x\} \rangle$ {S-extension}
  if t is not pruned **then**
    Compute $ES(t, D^p)$
    if $ES(t, D^p) \geq \theta$ **then**
      $L \leftarrow L \cup t$
      TraverseDFS(t, L_1)
    end if
  else
    Mark $\{x\}$ as invalid S-extension item.
  end if
  $t \leftarrow \langle s_1, \ldots, s_q \cup \{x\} \rangle$ {I-extension}
  if t is not pruned **then**
    Compute $ES(t, D^p)$
    if $ES(t, D^p) \geq \theta$ **then**
      $L \leftarrow L \cup t$
      TraverseDFS(t, L_1)
    end if
  else
    Mark $\{x\}$ as invalid I-extension item.
  end if
end for
return L
end function

6.1.3.1 Candidate Generation

Given a sequence s and the set of frequent 1-sequences L_1, we generate the S- and I-extensions of s as follows.

S-extension

We generate an S-extension t of s by appending an item $\{x\}$ as a new element to s. We first apply partial apriori pruning to t, say if t is a j-sequence, we first check to see if all the lexicographically smaller $(j-1)$-subsequences of t that would have already been explored in the mining process are also frequent. For example, if we are currently considering $t = \langle(d)(b)(c)\rangle$, a lexicographically smaller sub-sequence $\langle(b)(c)\rangle$ has already been explored, and by the apriori property if $\langle(b)(c)\rangle$ is not frequent, we do not need to consider $\langle(d)(b)(c)\rangle$.

Further, suppose that we S-extend a sequence s with an item $x_i \in L_1$ to obtain $t = \langle s\rangle \cdot \{x_i\}$ and suppose that t is not frequent. We then extend s with some other item $x_j \in L_1$ to obtain $t' = \langle s \cdot \{x_j\}\rangle$ and t' is frequent. We know by the apriori property that $\langle s \cdot \{x_j\} \cdot \{x_i\}\rangle$ can not be frequent either as it contains an infrequent sub-sequence $\langle s \cdot \{x_i\}\rangle$. Thus, if an extension of s using x_i is not frequent, we mark x_i as an invalid S-extension for s and do not consider extending any of the sequences for which s is a prefix with x_i. For example, for a sequence $\langle(d)\rangle$, if $\langle(d)(a)\rangle$ is not frequent, we mark $\{a\}$ as invalid S-extension, and say if $\langle(d)(b)\rangle$ is found to be frequent, we do not need to consider extending $\langle(d)(b)\rangle$ with $\{a\}$.

I-extension

We generate an I-extension t of s by appending an item $\{x\}$ to the last element in s. Similar to the S-extension case, we first apply partial apriori pruning to t. Then,

suppose that a sequence $s = \langle s_1, \ldots, s_q \rangle$ is I-extended with an item $x_i \in L_1$, i.e. $t = \langle s_1, \ldots, s_q \cup \{x_i\} \rangle$, and suppose that t is not frequent. We mark x_i as an invalid I-extension for s and do not consider extending any of the sequences for which s is a prefix with x_i. For example, for a sequence $\langle (a) \rangle$, suppose that $\{b\}$ is an invalid I-extension as $\langle (a, b) \rangle$ is not frequent then, we do not need extending $\langle (a)(c)(a) \rangle$ with $\{b\}$ either.

We now describe support computation for DFS.

6.1.3.2 Support Computation

Given a sequence s, the support computation task is to compute the expected support of s in the probabilistic database D^p. We propose two optimizations for this.

1. Observe that for a sequence t, where t is an S- or I-extension of s, for source i, if $\Pr[s \preceq D_i^p] = 0$, then $\Pr[t \preceq D_i^p] = 0$, i.e. if source i does not support s it cannot support t. When computing the expected support of s in D^p, we keep track of all the sources where $\Pr[s \preceq D_i^p] > 0$, denoted by $\mathcal{S}^s$, and when computing the expected support of t in D^p, we need only to visit the sources in $\mathcal{S}^s$.

2. Further, when doing the support computation for s, we save the $B_{i,s}$ array (recall Section 6.1.1) with every source $i \in \mathcal{S}^s$. When considering an S- or I-extension of s, we can use incremental support computation by reusing results from $B_{i,s}$. As the $B_{i,s}$ arrays are stored for all sources $i \in \mathcal{S}^s$ with each recursive call of DFS, in the worst case, a source may store up to j arrays, if s is a j-sequence.

6.2 Pattern Growth

We now give a pattern growth algorithm similar to PrefixSpan [40] for enumerating all frequent sequences. We first extend Definitions 2.11 and 2.12 to the probabilistic case.

Definition 6.4 (**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 6.5 (**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.

Recall from Chapter 2, Section 2.3.2, that the complete set of frequent sequential patterns could be partitioned into as many subsets as the number of frequent 1-sequences $\{\langle x_1 \rangle, \ldots, \langle x_n \rangle\}$, where 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 extend the definition of a projected database (Definition 2.13) to the probabilistic case.

Definition 6.6 (**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 $\{\langle(b, c : 0.7)(a, d, e : 0.3)\rangle, \langle(-, b : 0.9)(a, b, d : 0.8)(d, e : 0.3)\rangle\}$.

Recall from Section 2.3.2 that 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 6.4) and projected database (Definition 6.6), corresponding projected databases could be constructed and using the fast frequent 1-sequence computation (Section 6.1.1), 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, if an $\langle(a)\rangle$-projected database contains 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

Algorithm 5 Pattern-Growth Algorithm

Input: SLU probabilistic database D^p and support threshold θ.
Output: All sequences s with $ES(s, D^p) \geq \theta$.
$L_1 \leftarrow$ ComputeFrequent-1-sequences(D^p)
for all sequences $x \in L_1$ **do**
    Compute $B_{i,x}$ arrays
    Call ProjectedDB(x, $D^{s,p}$)
end for
function ProjectedDB(s, $D^{s,p}$)
$L_S \leftarrow$ Compute Frequent S-extensions
$L_I \leftarrow$ Compute Frequent I-extensions
Output all Frequent Sequences $\{s$ extended with x, for all x in L_S and $L_I\}$
for all $x \in L_S$ **do**
    $t \leftarrow \langle s \cdot \{x\}\rangle$ {S-extension}
    Compute $B_{i,t}$ arrays
    ProjectedDB(t, $D^{t,p}$)
end for
for all $x \in L_I$ **do**
    $t \leftarrow \langle s_1, \ldots, s_q \cup \{x\}\rangle$ {I-extension}
    Compute $B_{i,t}$ arrays
    ProjectedDB(t, $D^{t,p}$)
end for
end function

contribution – 0.75 – to the support of (b) in the projected database, but clearly their support for $\langle(a)(b)\rangle$ is different.

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.

6.2.1 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 source 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.

6.2.2 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.

6.2.2.1 S-extension Computation

We first compute the frequent S-extensions of s as follows.

TABLE 6.2: An example of computing the expected support of all S-extensions of $s = \langle(a)\rangle$ for D^p_X in the sample SLU database of Figure 4.3. The cells in columns labelled (i)–(iv) show the entries in F array after each event in D^p_X is processed. The values in the column (iv) are updated to G.

D^p_X	$(a, d, e : 0.6)$	$(b, c : 0.3)$	$(a, b, c : 0.7)$	$(c, d, e : 1.0)$
$B_{X,s}$	0.60	0.60	0.88	0.88
$\langle(a)\rangle$	0.000	0.000	0.420	0.420
$\langle(b)\rangle$	0.000	0.180	0.474	0.474
$\langle(c)\rangle$	0.000	0.180	0.474	0.880
$\langle(d)\rangle$	0.000	0.000	0.000	0.880
$\langle(e)\rangle$	0.000	0.000	0.000	0.880
	(i)	(ii)	(iii)	(iv)

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 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:

$$F[x] := ((1 - c_k) * F[x]) + (c_k * B_{i,s}[k - 1]). \tag{6.4}$$

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 6.2.

6.2.2.2 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:

$$F[x] := (1 - c_k) * F[x] + (B_{i,s}[k] - B_{i,s}[k-1] * (1 - c_k)). \tag{6.5}$$

We keep updating 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 $G[x] \geq \theta$ correspond to frequent I-extensions of s.

6.3 Summary

We have given probabilistic SPM algorithms for evaluating the interestingness predicate for expected support in an SLU database. We have considered two classes of classical sequential pattern mining algorithms. First, the candidate generation for which we have given optimizations for computing the frequent 1-sequences efficiently, for reusing the already computed DP matrix rows and for eliminating potential infrequent candidate sequences without expected support computation. We have considered a breadth-first and a depth-first exploration of the search space and have embedded the expected support computation sub-routine from Chapter 5 in our algorithms to yield probabilistic candidate generation algorithms. Finally, we have given a pattern growth algorithm by adapting the fast frequent 1-sequence computation and the DP sub-routine from Chapter 5 to work with projected databases. In the next chapter, we give an empirical evaluation of the optimizations and algorithms proposed in this chapter.

Chapter 7

Empirical Evaluation

We report on an empirical evaluation of the algorithms proposed in Chapter 6 (Section 7.3). We also evaluate the effectiveness of the probabilistic SPM framework in the presence of noise (Section 7.4).

In Chapter 6, we proposed probabilistic SPM algorithms based on the candidate generation (Section 6.1) and the pattern growth frameworks (Section 6.2). We proposed two search space exploration schemes based on the candidate generation framework, one based on a breadth-first exploration of the search space (BFS) and the other based on a depth-first exploration of the search space (DFS). We also proposed an approach based on the pattern growth framework (PGA). Further, we considered two ways of narrowing for BFS, and we also proposed probabilistic pruning both for BFS and DFS to eliminate potential infrequent candidate sequences without expected support computation (Section 6.1.1.4). In summary, we proposed:

1. BFS, that has two ways of narrowing which are Prefix based narrowing (PBN) and Hashtree based narrowing (HBN). Further, probabilistic pruning may also be used for BFS.

2. DFS, for which probabilistic pruning may also be used.

3. PGA.

In this chapter, we empirically evaluate these algorithms. We first describe the experimental setup, i.e. the platform, the datasets used and the SLU data generation. Then, we empirically evaluate PBN and HBN for BFS, and also the effectiveness of probabilistic pruning both for BFS and DFS (Section 7.2). Based on our empirical evaluation, we choose the most effective of the narrowing methods from PBN and HBN for BFS, and choose whether to use probabilistic pruning for BFS and DFS, or otherwise.

Thus, in our experiments we evaluate three algorithms, namely BFS, DFS and PGA, and perform the scalability analysis, i.e. we report the CPU time, memory usage and also the relative performance of our algorithms under different parameter settings using both real and synthetic datasets. Finally, we evaluate the effectiveness of the probabilistic SPM framework (Section 7.4), as we contrast the frequent sequences obtained from certain data to those obtained from uncertain data using both real and synthetic datasets.

7.1 Experimental Setup

In this section, we first describe the platform. Next, we describe the datasets used in our experiments and then discuss the SLU data generation. We then evaluate the effectiveness of the optimizations proposed in Section 6.1.1.

7.1.1 Platform

We have implemented our algorithms in C# (Visual Studio .Net 2010), and we have run our experiments on a machine with a 3.2GHz Intel CPU and 3GB RAM running Microsoft Windows XP (SP3). To obtain the results, i.e. running time, memory usage or others, we generate three probabilistic instances of each deterministic dataset, and run each of our algorithm on every probabilistic instance several times, and report the averages. In our implementations, we use some of the built-in features from Visual Studio .Net, e.g. the `Dictionary` class for efficiently locating candidate sequences in various operations.

We now give the datasets used in our experiments.

7.1.2 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 [27, 28, 31]. The real dataset *gazelle* is from Blue Martini Software, and was used for KDD-CUP'2000 [72]. 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. [72].

The synthetic datasets are generated using the IBM Quest data generator [12]. The list of parameters for the IBM Quest data generator is shown in Table 7.1 [73]. In our

TABLE 7.1: The list of parameters for the IBM Quest data generator.

IBM Quest Parameters	
Parameter	Meaning
N	Number of items
D	Number of source sequences
C	Average number of events per source sequence
T	Average number of items per itemset
N_I	Number of potential frequent itemsets
N_S	Number of potential frequent sequences
S	Average number of itemsets per frequent sequence
I	Average number of items per itemset in a frequent sequence

experiments, we fix the number of items $N = 2\text{K}$ 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 [36]: a dataset named `C`i`D`j`K` 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.

7.1.3 SLU Data Generation

As already mentioned that the real dataset `gazelle` and synthetic datasets from IBM Quest data generator are all deterministic datasets, and we transform these deterministic datasets to the probabilistic form [27, 28, 31]. 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 (Section 2.2). In an SLU database D^p, the source attribute σ is replaced by a probability distribution W over sources (Section 4.1.2.2).

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 σ_j in D^p as it was in D. While generating an SLU database, we set the number of sources in W to be at most two and as a result, the average source sequence length C is doubled in an SLU database D^p in contrast with the deterministic database D.

Specifically, while 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 σ_j as it was in D with a probability $1 - p$, and is associated with some other randomly chosen source $\sigma_k \in \mathcal{S}$, $k \neq j$, with probability p. Note that if the value of δ is low, there is still more certainty that r_i is associated with the same source σ_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 use support threshold θ as percentage of number of source sequences D rather than an absolute number $1 \leq \theta \leq m$ (Section 4.2.1) for convenience' sake.

We now demonstrate the effectiveness of the optimizations proposed in Chapter 6.

7.2 Effectiveness of Optimization

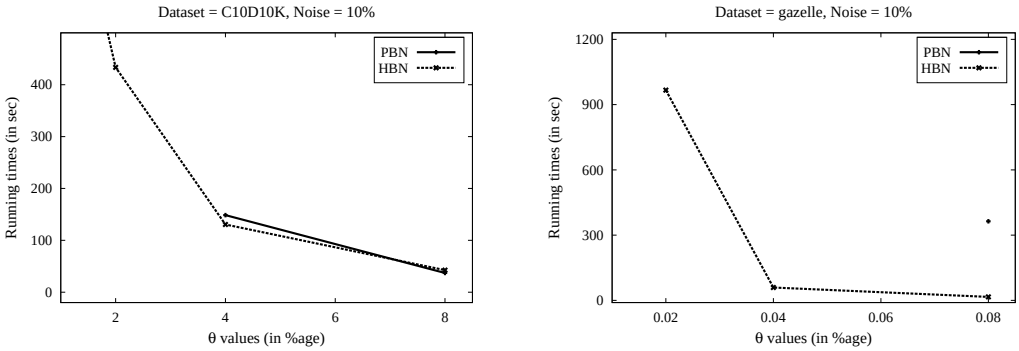
As already mentioned that We have considered two ways of narrowing for BFS namely PBN and HBN, we first study the effectiveness of these narrowing techniques. Then, we evaluate the effectiveness of probabilistic pruning.

7.2.1 Narrowing

Given the set of candidate j-sequences, recall that the objective in narrowing is to find a subset $N_{i,j}$ of the set of candidate j-sequences C_j that may be supported by source σ_i (Section 6.1.2). We considered two ways of narrowing. First is PBN where all the candidate j-sequences having a common $(j-1)$-prefix s' are stored as a single entry in the hashtable as a *(key, value)* pair where s' is the key and value is a pointer to the list of S- or I-extension of s'. Similarly in HBN, a hashtree is constructed using the candidate j-sequences and is used for narrowing.

We now show the effectiveness of the narrowing methods we consider. We consider two representative SLU datasets for the purpose which are `C10D10K` and `gazelle`, and report the running times for varying θ values in Figure 7.1.

We observe from the set of experiments in Figure 7.1 that not only is PBN relatively slower than HBN but that it has extensive memory requirements and runs out of memory for relatively harder instances, e.g. at $\theta = 2\%$ or less for `C10D10K`, or at $\theta = 0.04\%$ or less for `gazelle`, as indicated by missing dots in Figure 7.1.



(i) (ii)

FIGURE 7.1: Effectiveness of PBN and HBN. Missing points indicate that the algorithm went out of memory and thus, did not complete.

This can be understood as follows. As we demonstrate later in our experiments (Section 7.3) that under different parameter settings, e.g. for decreasing θ values, the number of frequent sequences increases rapidly. As storing candidate j-sequences based on a common $(j-1)$-prefix is not a very compact representation, the algorithm appears to require extensive memory when storing a large number of candidate sequences. HBN in contrast, uses a hashtree for narrowing and stores candidate sequences in a rather compact format and thus, scales better.

We conclude from the above set of experiments that HBN appears to be more scalable in contrast with PBN and thus, we only consider HBN for BFS in our experiments.

We now demonstrate the effectiveness of probabilistic pruning for the candidate generation approaches, namely BFS and DFS.

7.2.2 Probabilistic Pruning

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

and compute the upper bound as follows:

$$\Pr[s \preceq D_i^p] \leq \Pr[\langle s_1 \rangle \preceq D_i^p] * \ldots * \Pr[\langle s_q \rangle \preceq D_i^p], \tag{7.1}$$

whereas 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:

$$\Pr[s \preceq D_i^p] \leq \Pr[\langle s_1, \ldots, s_{q-1} \rangle \preceq D_i^p] * \Pr[\langle s_q \rangle \preceq D_i^p]. \tag{7.2}$$

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 Figure 7.2, 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 90% reduction by probabilistic pruning for infrequent candidate 2-sequences in most cases (all sub-figures of Figure 7.2 except (ii) and (iv)).

2. We also observe that probabilistic pruning is more effective for relatively harder instances, for example, for `C10D10K` at $\theta = 5\%$ vs. 10%.

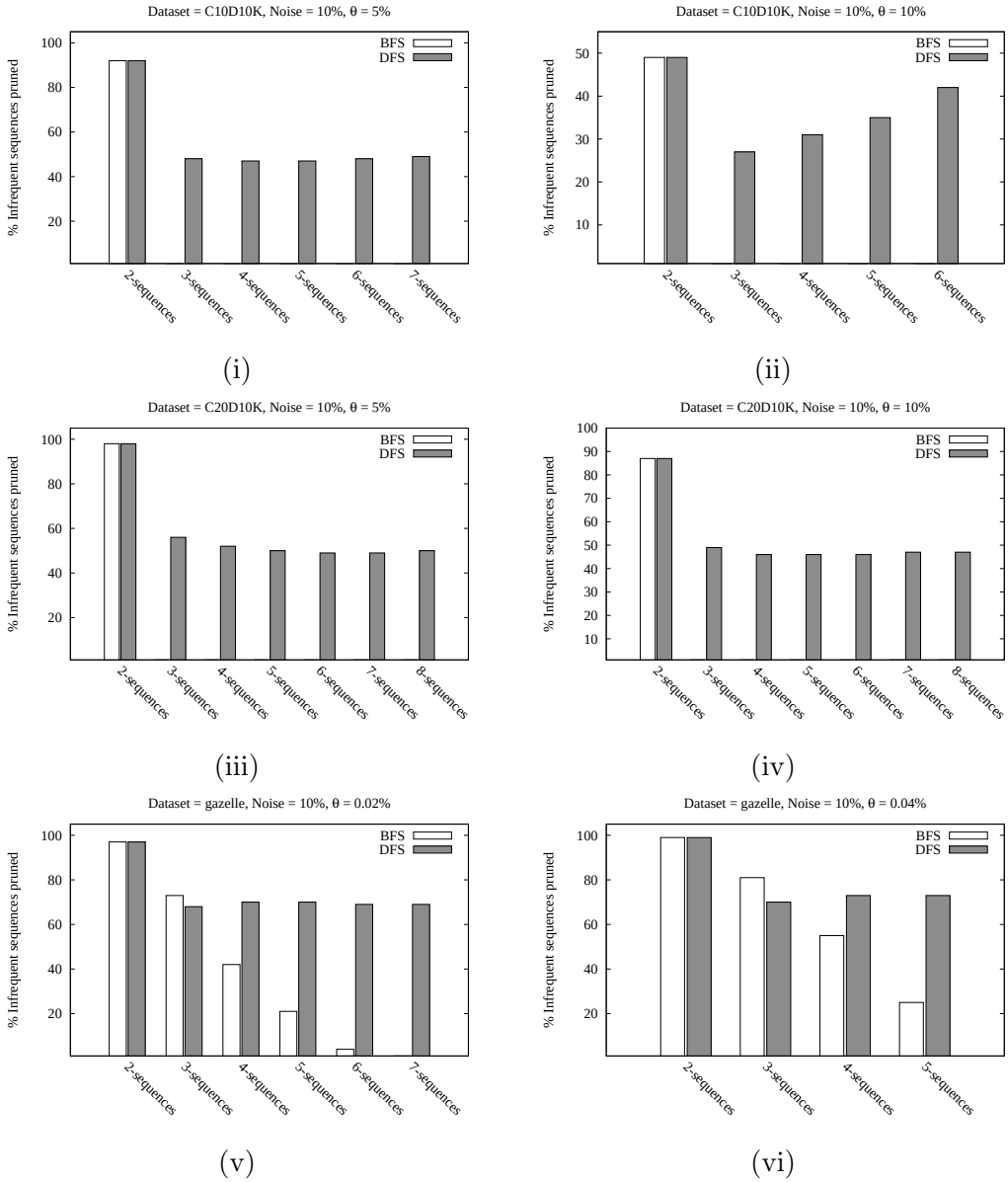


FIGURE 7.2: Effectiveness of Probabilistic Pruning for BFS and DFS. In these graphs, each bar indicates the percentage of infrequent candidate sequences eliminated by probabilistic pruning that passed apriori pruning.

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 (Equation 7.1 vs. Equation 7.2) 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 σ_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.

In summary, we select hashtree based narrowing and the probabilistic pruning for candidate 2-sequences only for BFS, and use probabilistic pruning for DFS for the entire duration of the algorithm.

We now demonstrate the scalability of probabilistic SPM algorithms.

7.3 Scalability Analysis

We evaluate the probabilistic SPM algorithms proposed in Chapter 6 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.

7.3.1 CPU Cost

We consider three parameters in our experiments, the support threshold θ, average number of events per source sequence C, and the number of source sequences D. Clearly, if the other two parameters are fixed, decreasing the θ values, or increasing the average number of events per source sequence C, or the number of source sequences D, all make an instance harder. For IBM Quest datasets, we test our algorithms against three parameter, namely for varying θ values, for increasing C, and for increasing D. 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`.

7.3.1.1 Varying θ

In the first set of experiments, Figure 7.3(i) and (ii), we test our algorithms for varying θ values for `gazelle`, and for a representative synthetic dataset `C10D10K`. We observe that the rate of increase in the running time for all algorithms is high for decreasing θ values.

To get an insight into this behaviour, we obtain the total number of frequent sequences for varying θ values in Figure 7.4(i), and also the distributions of frequent

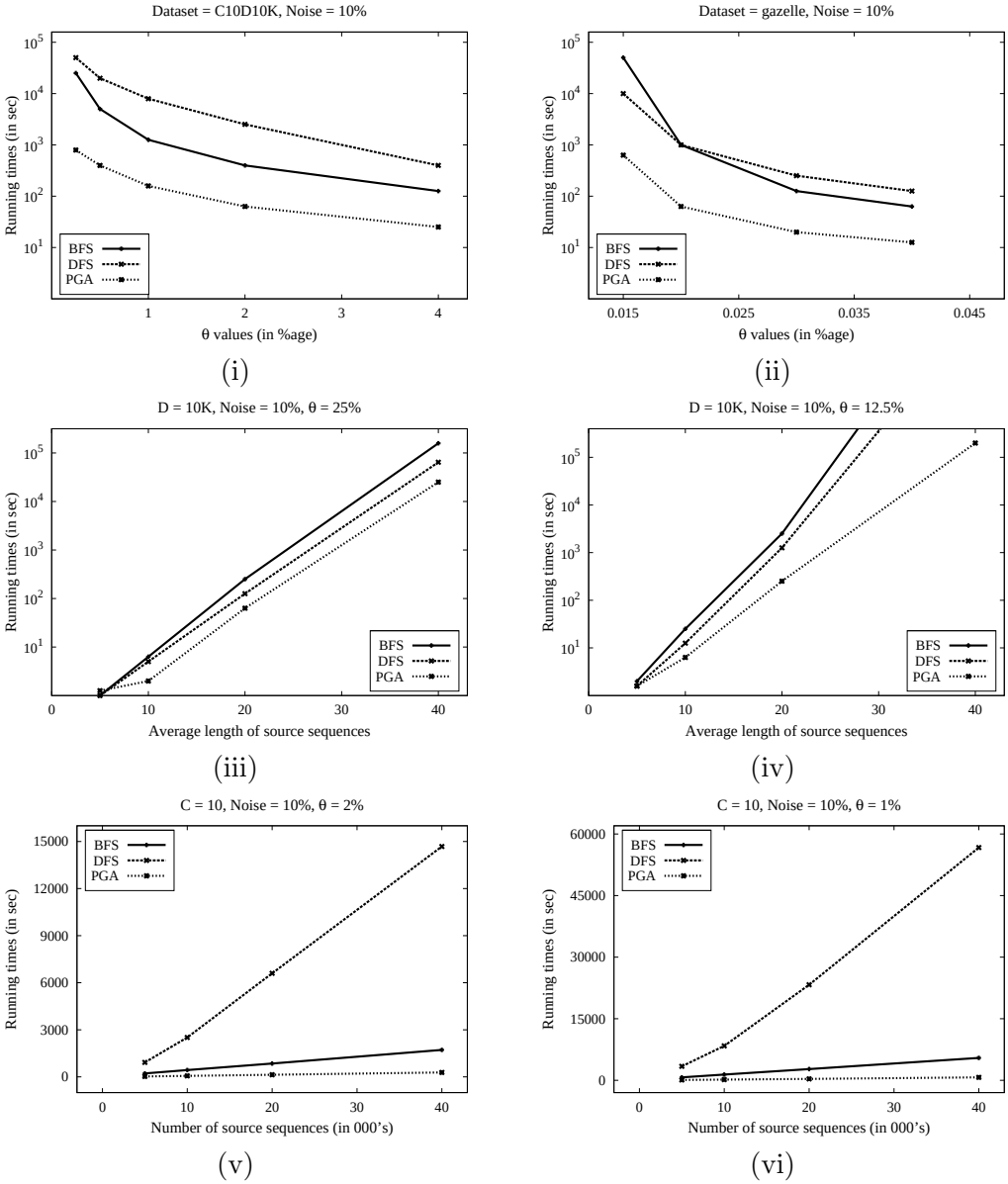


FIGURE 7.3: CPU time (in seconds) for BFS, DFS and PGA under different parameter settings for `gazelle` and representative synthetic datasets.

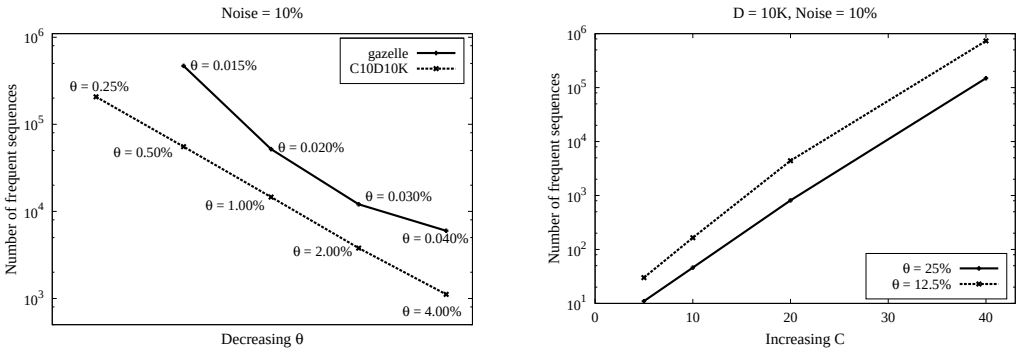


FIGURE 7.4: Number of frequent sequences, for varying θ values for `gazelle` and `C10D10K`, and for increasing C (Figure 7.3).

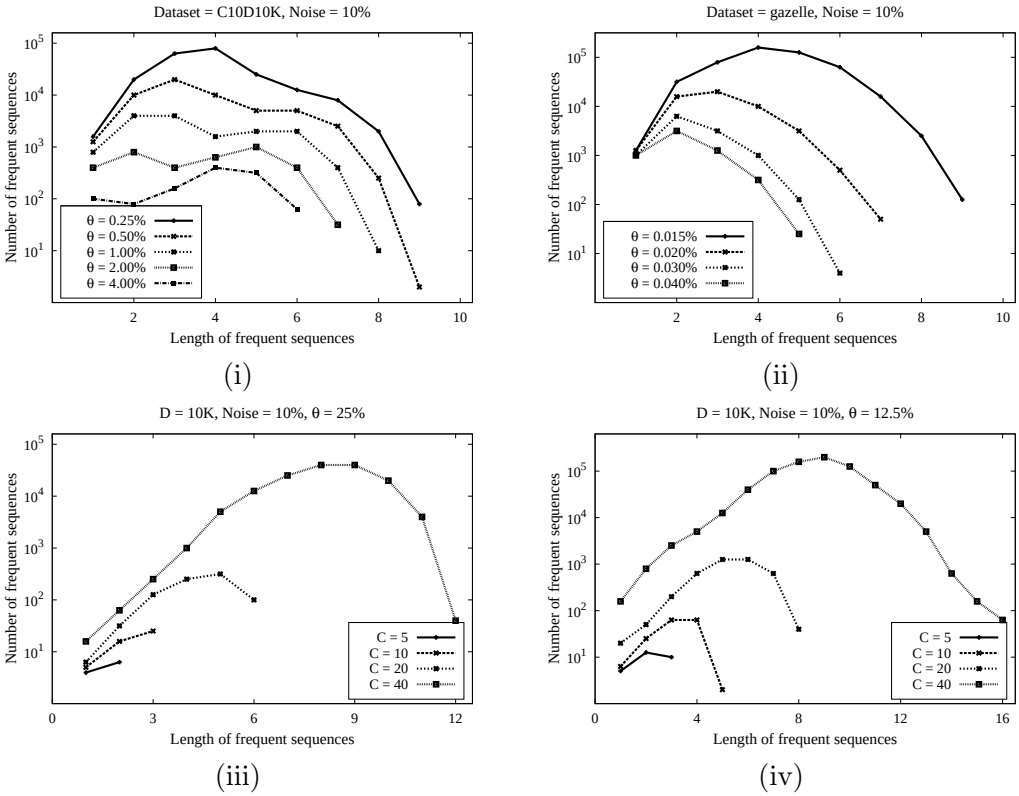


FIGURE 7.5: Distribution of frequent sequences for `gazelle` and representative synthetic datasets for the set of experiments in Figure 7.3(i)-(iv). As the distributions of frequent sequences for increasing D (Figure 7.3(v)-(vi)) remain relatively unaffected, we do not report those.

sequences in Figure 7.5(i) and (ii), for the set of experiments reported in Figure 7.3(i) and (ii).

We observe that decreasing θ results in a sharp increase in the number of frequent sequences (Figure 7.4(i)), and we can also see that the number of frequent sequences increase for sequences of all lengths (Figure 7.5(i) and (ii)) and consequently, we witness an increase in the running times for all algorithms.

7.3.1.2 Increasing C

In another set of experiments, Figure 7.3(iii) and (iv), 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 Figure 7.3(iii) and (iv) show that a linear increase in C results in an rapid increase in the running times for all algorithms. Similar to our earlier experiments for varying θ values, we obtain the total number of frequent sequences in Figure 7.4(ii), and the distributions of frequent sequences in Figure 7.5(iii) and (iv), for the set of experiments in Figure 7.3(iii) and (iv).

We observe that an increase in C results in rapid increase in the number of frequent sequences (Figure 7.4(ii)), and almost doubles the length of maximal frequent sequences (Figure 7.5(iii) and (iv)). Clearly, a rapid increase in the number of frequent sequences along with a two-fold increase in the length of maximal frequent sequences, makes an instance significantly harder which is also evident from the running time graphs.

7.3.1.3 Increasing D

We also test the scalability of our algorithms for increasing number of source sequences D, Figure 7.3(v) and (vi). 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 much greater than for BFS. Note that the distributions of frequent sequences or lengths of maximal frequent sequences remain almost unaffected as D increases, and we do not report those explicitly.

We now discuss the relative performance of our algorithms.

7.3.1.4 Comparison of Algorithms

We now contrast our algorithms in terms of CPU time for the set of experiments in Figure 7.3. We first focus on the candidate generation algorithms, BFS and DFS. We observe that while BFS is more efficient than DFS in CPU cost (Figure 7.3(i), (v), (vi)), DFS is better for increasing C (Figure 7.3(iii)-(iv)) 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, namely probabilistic pruning and support computation.

3. Probabilistic pruning helps DFS more than the BFS as shown in Figure 7.2.

Thus, depending on the interplay of these three factors, namely (1) number of candidate sequences generated (2) partial vs. 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 (Figure 7.3). In Figure 7.3(iv) 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.

7.3.2 Memory Usage

We also monitor the memory usage of our algorithms for the set of experiments in Figure 7.3, and report the peak memory used by each algorithm as a percentage of total system memory (RAM) in Figure 7.6.

- In Figure 7.6(i) and (ii), 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 Figure 7.6(iii) and (iv), 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 Figure 7.6(iv) as compared to Figure 7.6(iii) at a low θ, for $\theta = 12.5\%$ vs. 25%. We observe that the memory usage for PGA remains rather stable and increasing C (Figure 7.6(iii) and (iv)), or varying θ ($\theta = 25\%$ vs. 12.5%) does not significantly affect the memory usage of PGA.

- In Figure 7.6(v) and (vi), we observe that all the algorithms require relatively less memory as compared to the other experiments in Figure 7.6 (Figure 7.6(i)– (iv)). We observe that the memory usage for PGA almost doubles with a two-fold increase in the database size, for example for D = 20K vs. 40K in Figure 7.6(v) and (vi). We also observe that the memory usage of PGA is not affected by decrease in θ, Figure 7.6(v) and (vi) at $\theta = 2\%$ vs. 1%, similar to the observations in Figure 7.6(i)– (iv).

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,

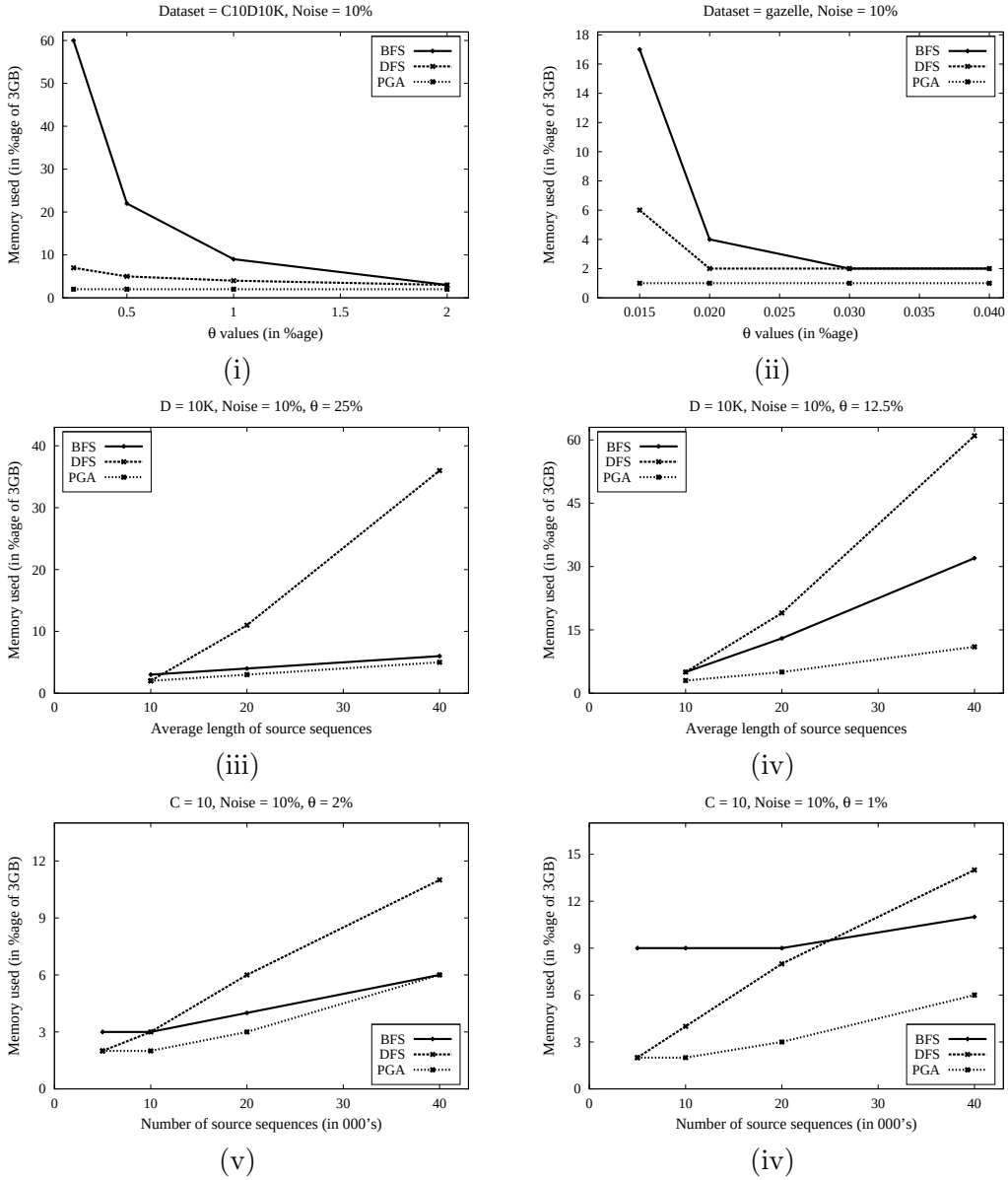


FIGURE 7.6: Memory usage (in terms of %age of system memory used) for the set of experiments in Figure 7.3.

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, while 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.

7.4 Effectiveness of Probabilistic SPM Framework

We now report on an evaluation of the probabilistic SPM framework in the presence of noise. Note that such studies have not been performed for probabilistic frequent itemset mining in literature [27, 28, 31]. As mentioned previously, the datasets used in our experiments are all deterministic datasets, and we artificially introduce uncertainty (noise) 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. We use the standard measures of *precision* and *recall* from information retrieval for the purpose [67, Chapter 8]. Precision and recall are set-based measures and 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 from a probabilistic dataset. Precision is defined as the

probability of a retrieved sequence being relevant:

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

whereas recall is defined as the probability of a relevant sequence being retrieved:

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

For example, say if $|R| = 100$, i.e. the number of frequent sequences retrieved from a deterministic dataset is 100, $|R'| = 80$, i.e. the number of frequent sequences retrieved from a noisy dataset is 80, and $|R \cap R'| = 40$, i.e. only 40 of the retrieved 80 frequent sequences are in the 100 frequent sequences retrieved from the deterministic dataset, then the precision is $40/80 = 0.50$, and the recall is $40/100 = 0.40$.

We report on an evaluation of the probabilistic SPM framework in terms of precision and recall using both real and synthetic datasets. We report *overall* precision and recall for the complete set of frequent sequences, as well as for 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.

7.4.1 Synthetic Datasets

In the first set of experiments, Table 7.2, 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 experiment, Table 7.3, we consider the dataset `C20D10K` at $\theta = 20\%$ and 10%, and report the precision and recall results similar to Table 7.2.

TABLE 7.2: 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

We observe that for our considered datasets, namely C10D10K and C20D10K and for our considered θ values, change in C or θ, does not considerably affect precision/recall. However, when δ is varied (between 5% to 20% in our experiments), both precision and recall are effected. Thus, whilst precision is slightly worse than recall when δ is increased, we get over 90% overall recall in our experiments in Tables 7.2 and 7.3. Further, we get over 80% overall precision as well except for C20D10K for $\theta = 20\%$ and $\delta = 20\%$, where it is 77% (Table 7.3).

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 $\theta = 20\%$.

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 $\theta = 20\%$ vs. $\theta = 10\%$.

We conclude from the set of experiments in Tables 7.2 and 7.3 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.

7.4.2 `gazelle`

We now evaluate the effectiveness of probabilistic SPM framework for `gazelle`. In our experiments (Table 7.4), 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 7.4, even the

TABLE 7.3: 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

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

TABLE 7.4: 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*

downwards in order to improve recall. 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 that 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 7.5 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

TABLE 7.5: The updated precision and recall results for `gazelle` after θ is revised. The rows labelled as 'Change' show the improvement/decline (+/-) in the precision and recall results 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*

rows show the updated precision and recall values.

We report the updated precision and recall results after revising θ in Table 7.5. 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 $\delta = 20\%$, although 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 that 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.

In the empirical evaluation of probabilistic SPM, some promising results have been shown and clearly more work needs to be done.

(1) Our empirical evaluation is not a validation of the uncertainty model that we consider, rather we have demonstrated that if SLU data is available, the probabilistic SPM framework gives promising results. The validation of the uncertainty model using real-life data is something to be considered in future research.

(2) For all our experiments, we have only considered a range of parameter values for parameters like W, θ, δ, etc. It is rather obvious that choosing a different range of parameter values may have an impact on the results; as already seen for some parameter settings in our experiments.

(3) We have considered a few synthetic datasets but only one real dataset. The performance of the probabilistic SPM framework for other real datasets may well also be different.

(4) The manner in which uncertainty is introduced into the datasets is also important. The transformation from deterministic data to the SLU data was done without knowing the time-stamps (IBM Quest datasets or `gazelle` do not have time-stamps); the informations about the time-stamps may also have an impact on the SLU data and consequently, the results.

7.5 Summary

In this chapter, we have demonstrated the effectiveness of the optimizations, namely narrowing and probabilistic pruning, and the scalability of the algorithms, namely BFS, DFS and PGA, proposed in Chapter 6 with the help of experiments. Thus, we have shown that whilst the performance of BFS and DFS depends on individual parameter settings and is not better than PGA for our considered sets of experiments, PGA is the most scalable approach in terms of CPU time and the memory usage. We have also evaluated the effectiveness of probabilistic SPM framework in the presence of noise, and we have shown that whilst we get encouraging results in case of synthetic datasets, the expected support threshold needs to be fine tuned in order to obtain better results for `gazelle`. In the end, we have discussed that although

we obtain promising results for the sets of experiments we consider, a number of open directions remain to be explored in future.

Chapter 8

Conclusions and Future Work

The main objective of this thesis was to study the sequential pattern mining (SPM) problem in probabilistic databases. Although there has been prior work on SPM in uncertain data, this thesis is the first to study SPM in the context of probabilistic databases. Probabilistic databases are a relatively recent but very influential framework for modelling uncertain data. As this is the first study of SPM in probabilistic databases, the focus has been more on looking at the basic SPM problem, and understanding the foundational issues. Indeed, the formalization of the SPM problem in probabilistic databases is itself not obvious.

8.1 Our Contributions

We summarize our main contributions (based on the research questions in Section 1.2) below:

(1) We first formalized the probabilistic SPM problem in probabilistic databases, and we proposed four uncertainty models, namely TLU, ELU, SLU and SLU-D. Then, we defined the interestingness predicate based on two interestingness

measures, namely expected support and probabilistic frequentness, for the SPM problem in probabilistic databases. The formulations were motivated by real life examples, thus establishing that SPM is probabilistic databases is an interesting topic of study. Indeed, our work has been followed up by Wan [74], Zhao et al. [46] and Gupta et al. [75].

(2) We discussed evaluating the interestingness predicate from a complexity-theoretic viewpoint, and we showed that different uncertainty models have different outcomes from a complexity theoretic viewpoint under different interestingness measures. We showed that whilst it is NP-complete to evaluate the interestingness predicate for computing expected support for the SLU-D model, we show that it can be evaluated in polynomial time for the TLU, ELU and SLU models by giving a DP algorithm. We also showed that whilst the interestingness predicate for computing probabilistic frequentness can be evaluated in polynomial time for the TLU and ELU models, it is #P-complete to do the same for the SLU model. We note that models and measures that we propose that try to capture dependencies between tuples tend to be intractable in general.

(3) We considered the SLU model and the expected support measure, and extended classical SPM algorithms based on candidate generation and pattern growth frameworks to work under probabilistic settings. Thus, we proposed two candidate generation algorithms based on a breadth-first and a depth-first exploration of the search space, namely BFS and DFS, and a pattern growth algorithm, namely PGA based on the idea of projected databases. We also proposed optimizations like fast frequent 1-sequence computation, incremental support computation and probabilistic pruning, to speed up the support computation task.

(4) We gave an empirical evaluation to demonstrate the usefulness of the optimizations we proposed and also the scalability of probabilistic SPM algorithms that we proposed, and the results show that optimized SPM is scalable and can be done in reasonable time. We also demonstrated the effectiveness of the probabilistic SPM framework for the SLU model using the expected support measure.

8.2 Future Work

As this is a first step towards mining sequential patterns in probabilistic databases, there are a number of directions to consider in future.

- In this thesis, our implementations are for the SLU model for the expected support measure. It would be interesting to compute a more informative measure such as probabilistic frequentness, for the TLU or ELU models. Further, an approximate solution for probabilistic frequentness, as already proposed for frequent itemset mining [29], could also be an interesting future work.

- In addition to the interestingness measures we consider, namely expected support and probabilistic frequentness, other interestingness measures, e.g. expected rank [3] have also been proposed in literature. It would be interesting to compute these measures for different uncertainty models and then contrasting the results obtained from different interestingness measures.

- In the uncertainty models we consider, we consider uncertainty either in the tuple or in the attribute. Considering a more expressive uncertainty model, e.g. x-tuples, also seems a potential challenge.

- We have implemented a relatively simple SLU data generator in our experiments. A more realistic data generator which is able to generate some kind of dependencies in data also seems a potential future work.

- We have only considered mining the set of sequential patterns. Mining maximal/closed sequences could also be a future work. However, even the definition of a closed sequential pattern is not clear in the probabilistic case.

- A variety of real-world applications have been proposed for classical SPM. Zhao et al. [46] have recently proposed RFID trajectory mining using probabilistic SPM. However, there is a need for more real-world applications for probabilistic SPM and looking for further real-world applications for probabilistic SPM can also be a potential future work.

Appendix A

Complexity Classes

We give an overview of complexity classes. We first discuss complexity classes P, NP, and #P, followed by the idea of polynomial time reducibility leading to the notion of NP-completeness and #P-completeness.

We first focus on decision problems which are problems for which the answer is simply yes or no. For example:

- in the boolean satisfiability problem (SAT), the problem instance is a boolean formula. Given a boolean formula, is it satisfiable i.e. is it possible to assign true or false values to the variables in the formula such that the boolean formula evaluates to true?
- in an undirected graph, a Hamiltonian path is a path that visits each vertex exactly once. Given an undirected graph, is it possible to find a Hamiltonian path?

The class P is the class of problems that can be solved in polynomial time. Specifically, given an input of size n, the problems in P can be solved in time $O(n^k)$ where

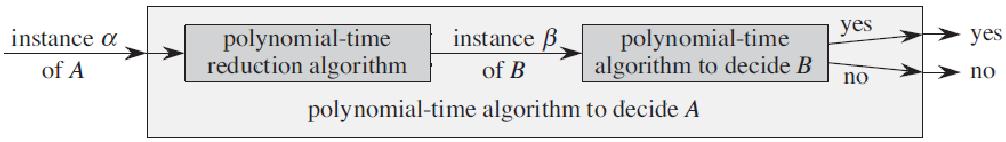


FIGURE A.1: We transform an instance α of A in polynomial time to an instance β of B. We solve β in polynomial time and the answer to β is the answer to α (image from Cormen et al. [4]).

k is a constant. The problems in P are considered computationally feasible. The class NP is the class of problems for which a certificate (solution) can be verified in polynomial time. A problem is NP-complete if it is in NP and is as hard as any other problem in NP.

A problem in P is also in NP: if a problem can be solved in polynomial time, a solution to a problem can also be verified in polynomial time. Thus, it is clear that P $\subseteq$ NP but whether P $\subset$ NP is still open problem, and is one of the most important problems in computer science. The question whether P $\subset$ NP is important because the best known exact solutions to NP-complete problems take exponential time in the worst cases and are therefore currently computationally intractable.

Another important complexity class #P is the class of counting problems for which we are interested in how many 'yes' instances there are for a decision problem (the decision problem is either in P or NP). Similar to the class NP-complete, a problem is #P-complete if it is in #P and is as hard as any other problem in #P.

A.1 Reducibility

We now discuss the idea of reducibility. Suppose that we want to solve a decision problem A in polynomial time, and that we are given the input α which is an instance of A. Now suppose that we already know how to solve a decision problem

B in polynomial time. Finally, suppose that we have a sub-routine that transforms an instance α of A into an instance β of B such that the following conditions hold:

1. The transformation from α to β is polynomial time.
2. The answer to α is yes if and only if the answer to β is yes, and the answer to α is no if and only if the answer to β is no.

Figure A.1 illustrates the idea of reducibility, the outline of which is as follows:

1. Given an instance α of A, transform α to an instance β of B using a polynomial time reduction algorithm.
2. Execute the polynomial time decision algorithm for B on β.
3. The answer for β is the same as the answer for α.

If each of the above three steps can be performed in polynomial time then we can decide on α in polynomial time.

However, for NP-completeness, we do not want to show that problem A is as easy as B. We are rather interested in determining how hard a problem is. We use the polynomial time reductions the opposite way for this. Suppose that we want to show that no polynomial time algorithm can exist for a particular decision problem B, and that we have a decision problem A for which we already know that no polynomial time algorithm can exist. Further, we have a polynomial time reduction that transforms an instance of A to an instance of B. We can show that no polynomial time algorithm can exist for B by using contradiction. Suppose that B can be solved in polynomial time. Then, using the method shown in Figure A.1, we have a way to solve problem A in polynomial time, which contradicts our assumption that there is no polynomial time algorithm for A.

We now formally define the complexity classes P, NP and NP-complete, for which we first discuss the notion of a decision problem and problem encoding.

A.2 Decision Problems

As the theory of NP-completeness is focussed on decision problems, which are the problems having a yes/no solution, we first formally define the notion of a decision problem. We define a *decision problem* Q as a function that maps the instance set I to the solution set $\{0, 1\}$.

A.2.1 Problem Encoding

An *encoding* of a set S of objects is a mapping e from S to the set of binary strings. It is important to note that any decision problem can be encoded as a binary string. We define a *concrete decision problem* as a problem that has its instance set as a set of binary strings.

Given a decision problem Q mapping an instance set I to $\{0, 1\}$, an encoding $e : I \to \{0, 1\}^*$ can be used to induce a related concrete decision problem, which we denote by $e(Q)$. If the solution to a decision problem $i \in I$ is $Q(i) \in \{0, 1\}$, then the solution to the concrete decision problem instance $e(i) \in \{0, 1\}^*$ is also $Q(i)$.

Thus, a computer algorithm that solves a decision problem actually takes an encoding of a problem instance as input. We now define the complexity class P.

Definition A.1 (P). The complexity class P is the class of concrete decision problems that are polynomial time solvable.

Remark A.2. The choice of encoding is crucial in understanding the concept of polynomial time solvable from concrete decision problems to decision problems. For

example, suppose that the only input to an algorithm is an integer k, and suppose that the time complexity of the algorithm is $O(k)$. Now assume that the integer k is given as a unary number — a string of k 1's — then for an input of length n, the algorithm completes in $O(n)$ time. Now consider the binary representation of integer k, then the input of length $n = \lfloor \log k \rfloor + 1$. In this case, the time complexity of the algorithm is $O(k) = O(2^n)$, which is exponential in the size of the input. Thus, an algorithm runs in either polynomial or superpolynomial time depending on the encoding. In this thesis, we assume a binary encoding for any integers present in the input.

A.2.2 NP and NP-completeness

As we focus on decision problems, we can use concepts from formal language theory. Given an alphabet $\{0, 1\}$, a *language* L over $\{0, 1\}$ is any set of strings made of symbols in $\{0, 1\}$. Thus, a decision problem Q as a language L over the alphabet $\{0, 1\}$ can be defined as follows: $L = \{x \in \{0, 1\}^* : Q(x) = 1\}$. We now discuss the class NP using the notion of language.

The complexity class NP is the class of languages that can be verified by a polynomial time algorithm. Formally, a language L belongs to NP if and only if there exists a polynomial time algorithm A and a constant c such that:

$$L = \{x \in \{0, 1\}^* : \exists \text{ a string } y \text{ with } |y| = O(|x|^c) \text{ such that } A(x, y) = 1\}$$

The string y in the above definition is called a *certificate*. Moreover, if $L \in$ P, then $L \in$ NP, since if there is a polynomial time algorithm to decide L, that algorithm can easily be converted to a verification algorithm that simply ignores any certificate and accepts exactly those input strings it determines to be in L. Thus, P $\subseteq$ NP.

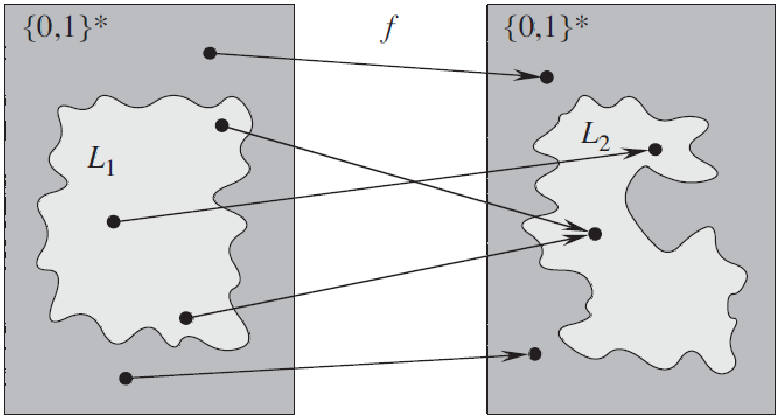


FIGURE A.2: An illustration of a polynomial time reduction from a language L_1 to a language L_2 using a reduction function f (image from Cormen et al. [4]).

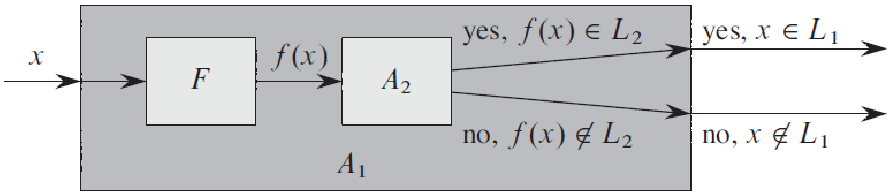


FIGURE A.3: An algorithm A_1 that decides whether $x \in L_1$ by using F to transform x to $f(x)$ and then using A_2 to decide whether $f(x) \in L_2$ (image from Cormen et al. [4]).

A problem Q can be reduced to another problem Q' if any instance of Q can be transformed to an instance of Q', the solution to which provides a solution to the instance of Q.

We say that a language L_1 is polynomial time reducible to a language L_2, denoted by $L_1 \leq_P L_2$, if there exists a polynomial time computable function $f : \{0,1\}^* \to \{0,1\}^*$ such that for all $x \in \{0,1\}^*$, $x \in L_1$ if and only if $f(x) \in L_2$. Figures A.2 and A.3 illustrate the idea of a polynomial time reduction from a language L_1 to another language L_2. Each language is a subset of $\{0,1\}^*$. The reduction function f provides a polynomial time mapping such that if $x \in L_1$, then $f(x) \in L_2$. Moreover, if $x \notin L_1$, then $f(x) \notin L_2$. The answer to the question, whether $f(x) \in L_2$ is directly the answer to whether $x \in L_1$.

Definition A.3 (NP-complete). A language $L \subseteq \{0,1\}^*$ is *NP-complete* if the following holds: (1) $L \in NP$ and (2) $L' \leq_P L$ for every $L' \in$ NP.

In other words, Definition A.3 means that if any of the NP-complete problems can be solved in polynomial time, all of them can be solved in polynomial time. If a language L satisfies property 2, but not necessarily property 1, we say that L is *NP-hard*.

We have formally defined the notion of classes P, NP and NP-complete. We have also discussed that why it is important that an instance of a problem be encoded as binary. We have also illustrated the concept of polynomial time reducibility to show the hardness of the problems. A more detailed discussion can be found in Cormen et al. [4, Chapter 34].

We now focus on the class #P.

A.3 Counting Problems: The Class #P

As already mentioned that the class #P is the class of problems for which we want to know how many solutions exist to a problem. For example, for the SAT and Hamiltonian Path path problems discussed for class NP, the corresponding #P problems are:

- #SAT: that is, given a boolean formula, compute the number of different truth assignments that satisfy it.
- #Hamiltonian Path: that is, given an undirected graph, compute the number of different Hamiltonian paths in the graph.

Note that if the counting version of a problem in NP can be solved, it is sufficient to get an answer to the decision version of the problem. For example, the answer to #SAT is sufficient to answer SAT. Thus, the counting version is considered somewhat harder than the corresponding decision version.

We view languages as binary relations such that a problem is specified by a binary relation $Q = \{(x, y)\}$, where x is the encoding of an input and y is the encoding of the solution. A binary relation is *polynomially balanced* if for each input instance x, the only possible solutions y have length at most $|x|^k$, and the alphabet of y is $\{0, 1\}$.

Definition A.4 (#P). Let Q be a polynomially balanced, (non-deterministic) polynomial time decidable binary relation. The counting problem associated with Q is the following: Given x, how many y are there such that $(x, y) \in Q$? (The output is an integer in binary).

Definition A.5 (#P-complete). A relation $Q \subseteq \{0, 1\}^*$ is *#P-complete* if (1) $Q \in$ #P and (2) for every $Q' \in$ #P, Q' can be reduced to Q in polynomial time.

If a relation Q satisfies property 2, but not necessarily property 1, we say that Q is #P-hard.

Note that the polynomial time reduction, reducing Q' to Q must be such that given an x (an encoding of an instance of Q'), the reduction $f(x)$ should map x to the encoding of an instance of Q in such a way that counting the number of pairs $(f(x), y) \in Q$ allows us to compute the number of pairs $(x, y) \in Q'$. Reductions for proving #P-completeness are usually (but not always) *parsimonious*: the number of pairs $(f(x), y) \in Q$ is exactly equal to the number of pairs $(x, y) \in Q'$, i.e. the number of solutions is preserved by the reduction.

The most interesting #P-complete problems are those for which the corresponding decision problem can be solved in polynomial time. For example, the problem of

counting the perfect matchings in a bipartite graph (used in Chapter 5). See Papadimitriou [76, Chapter 18] for more details.

Bibliography

[1] Nilesh N. Dalvi, Christopher Ré, and Dan Suciu. Probabilistic databases: diamonds in the dirt. *Communications of the ACM*, 52(7):86–94, 2009.

[2] O. Hassanzadeh and R. J. Miller. Creating probabilistic databases from duplicated data. *The VLDB Journal*, 18(5):1141–1166, 2009.

[3] Graham Cormode, Feifei Li, and Ke Yi. Semantics of ranking queries for probabilistic data and expected ranks. In *ICDE*, pages 305–316. IEEE, 2009. ISBN 978-0-7695-3545-6.

[4] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms (3rd edition)*. MIT Press, 2009. ISBN 978-0-262-03384-8.

[5] Usama M. Fayyad, Gregory Piatetsky-Shapiro, and Padhraic Smyth. From data mining to knowledge discovery in databases. *AI Magazine*, 17(3):37–54, 1996.

[6] Usama M. Fayyad, Gregory Piatetsky-Shapiro, and Padhraic Smyth. The KDD process for extracting useful knowledge from volumes of data. *Communications of the ACM*, 39(11):27–34, 1996.

[7] Jiawei Han, Micheline Kamber, and Jian Pei. *Data Mining: Concepts and Techniques*. Morgan Kaufmann, second edition, 2005.

[8] Usama M. Fayyad, Gregory Piatetsky-Shapiro, Padhraic Smyth, and Ramasamy Uthurusamy, editors. *Advances in Knowledge Discovery and Data Mining*. AAAI/MIT Press, 1996. ISBN 0-262-56097-6.

[9] Pang-Ning Tan, Vipin Kumar, and Jaideep Srivastava. Selecting the right interestingness measure for association patterns. In *KDD* DBL [82], pages 32–41. ISBN 1-58113-567-X.

[10] Craig Silverstein, Sergey Brin, and Rajeev Motwani. Beyond market baskets: Generalizing association rules to dependence rules. *Data Mining and Knowledge Discovery*, 2(1):39–68, 1998.

[11] Rakesh Agrawal, Tomasz Imielinski, and Arun N. Swami. Mining association rules between sets of items in large databases. In Peter Buneman and Sushil Jajodia, editors, *SIGMOD Conference*, pages 207–216. ACM Press, 1993.

[12] Rakesh Agrawal and Ramakrishnan Srikant. Mining sequential patterns. In Philip S. Yu and Arbee L. P. Chen, editors, *ICDE*, pages 3–14. IEEE Computer Society, 1995. ISBN 0-8186-6910-1.

[13] Manish Gupta and Jiawei Han. Applications of pattern discovery using sequential data mining. In Pradeep Kumar, P. Radha Krishna, and S. Bapi Raju, editors, *Pattern Discovery Using Sequence Data Mining: Applications and Studies*, chapter 1, pages 1–23. IGI Global, 2012.

[14] Manish Gupta and Jiawei Han. Approaches for pattern discovery using sequential data mining. In Pradeep Kumar, P. Radha Krishna, and S. Bapi Raju, editors, *Pattern Discovery Using Sequence Data Mining: Applications and Studies*, chapter 8, pages 137–154. IGI Global, 2012.

[15] Nizar R. Mabroukeh and Christie I. Ezeife. A taxonomy of sequential pattern mining algorithms. *ACM Computing Surveys*, 43(1):3, 2010.

[16] Bamshad Mobasher, Honghua Dai, Tao Luo, and Miki Nakagawa. Discovery and evaluation of aggregate usage profiles for web personalization. *Data Mining and Knowledge Discovery*, 6(1):61–82, 2002.

[17] Simon Jaillet, Anne Laurent, and Maguelonne Teisseire. Sequential patterns for text categorization. *Intelligent Data Analysis*, 10(3):199–214, 2006.

[18] Themis P. Exarchos, Costas Papaloukas, Christos Lampros, and Dimitrios I. Fotiadis. Mining sequential patterns for protein fold recognition. *Journal of Biomedical Informatics*, 41(1):165–179, 2008.

[19] Charu C. Aggarwal, editor. *Managing and Mining Uncertain Data*. Springer, 2009.

[20] Nodira Khoussainova, Magdalena Balazinska, and Dan Suciu. Probabilistic event extraction from RFID data. In *ICDE* DBL [78], pages 1480–1482.

[21] Jennifer Widom. TRIO: A system for data, uncertainty, and lineage. In Charu C. Aggarwal, editor, *Managing and Mining Uncertain Data*, chapter 5. Springer, 2009.

[22] Charu C. Aggarwal and Philip S. Yu. A survey of uncertain data algorithms and applications. *IEEE Transactions on Knowledge and Data Engineering*, 21 (5):609–623, 2009.

[23] Charu C. Aggarwal. On unifying privacy and uncertain data models. In *ICDE* DBL [78], pages 386–395.

[24] Dan Suciu and Nilesh N. Dalvi. Foundations of probabilistic answers to queries. In Özcan [81], page 963. ISBN 1-59593-060-4.

[25] Liwen Sun, Reynold Cheng, David W. Cheung, and Jiefeng Cheng. Mining uncertain data with probabilistic guarantees. In Bharat Rao, Balaji Krishnapuram, Andrew Tomkins, and Qiang Yang, editors, *KDD*, pages 273–282. ACM, 2010. ISBN 978-1-4503-0055-1.

[26] Carson Kai-Sang Leung, Mark Anthony F. Mateo, and Dale A. Brajczuk. A tree-based approach for frequent pattern mining from uncertain data. In Washio et al. [80], pages 653–661. ISBN 978-3-540-68124-3.

[27] Charu C. Aggarwal, Yan Li, Jianyong Wang, and Jing Wang. Frequent pattern mining with uncertain data. In Elder et al. [77], pages 29–38. ISBN 978-1-60558-495-9.

[28] Thomas Bernecker, Hans-Peter Kriegel, Matthias Renz, Florian Verhein, and Andreas Züfle. Probabilistic frequent itemset mining in uncertain databases. In Elder et al. [77], pages 119–128. ISBN 978-1-60558-495-9.

[29] Toon Calders, Calin Garboni, and Bart Goethals. Approximation of frequentness probability of itemsets in uncertain data. In Geoffrey I. Webb, Bing Liu, Chengqi Zhang, Dimitrios Gunopulos, and Xindong Wu, editors, *ICDM*, pages 749–754. IEEE Computer Society, 2010.

[30] Dimitrios Gunopulos, Roni Khardon, Heikki Mannila, Sanjeev Saluja, Hannu Toivonen, and Ram Sewak Sharm. Discovering all most specific sentences. *ACM Transactions on Database Systems*, 28(2):140–174, 2003.

[31] Chun Kit Chui, Ben Kao, and Edward Hung. Mining frequent itemsets from uncertain data. In Zhi-Hua Zhou, Hang Li, and Qiang Yang, editors, *PAKDD*, volume 4426 of *Lecture Notes in Computer Science*, pages 47–58. Springer, 2007. ISBN 978-3-540-71700-3.

[32] Jiawei Han, Hong Cheng, Dong Xin, and Xifeng Yan. Frequent pattern mining: current status and future directions. *Data Mining and Knowledge Discovery*, 15(1):55–86, 2007.

[33] Jochen Hipp, Ulrich Güntzer, and Gholamreza Nakhaeizadeh. Algorithms for association rule mining - a general survey and comparison. *SIGKDD Explorations*, 2(1):58–64, 2000.

[34] Jay Ayres, Jason Flannick, Johannes Gehrke, and Tomi Yiu. Sequential pattern mining using a bitmap representation. In *KDD* DBL [82], pages 429–435. ISBN 1-58113-567-X.

[35] Xifeng Yan, Jiawei Han, and Ramin Afshar. Clospan: Mining closed sequential patterns in large databases. In Daniel Barbará and Chandrika Kamath, editors, *SDM*. SIAM, 2003. ISBN 0-89871-545-8.

[36] Mohammed Javeed Zaki. SPADE: An efficient algorithm for mining frequent sequences. *Machine Learning*, 42(1/2):31–60, 2001.

[37] Guizhen Yang. Computational aspects of mining maximal frequent patterns. *Theoretical Computer Science*, 362(1-3):63–85, 2006.

[38] Ramakrishnan Srikant and Rakesh Agrawal. Mining sequential patterns: Generalizations and performance improvements. In Peter M. G. Apers, Mokrane Bouzeghoub, and Georges Gardarin, editors, *EDBT*, volume 1057 of *Lecture Notes in Computer Science*, pages 3–17. Springer, 1996. ISBN 3-540-61057-X.

[39] Jiawei Han, Jian Pei, Behzad Mortazavi-Asl, Qiming Chen, Umeshwar Dayal, and Meichun Hsu. Freespan: frequent pattern-projected sequential pattern mining. In *KDD*, pages 355–359, 2000.

[40] Jian Pei, Jiawei Han, Behzad Mortazavi-Asl, Jianyong Wang, Helen Pinto, Qiming Chen, Umeshwar Dayal, and Meichun Hsu. Mining sequential patterns by

pattern-growth: The PrefixSpan approach. *IEEE Transactions on Knowledge and Data Engineering*, 16(11):1424–1440, 2004.

[41] Unil Yun. A new framework for detecting weighted sequential patterns in large sequence databases. *Knowledge-Based Systems*, 21(2):110–122, 2008.

[42] Minos N. Garofalakis, Rajeev Rastogi, and Kyuseok Shim. Spirit: Sequential pattern mining with regular expression constraints. In Malcolm P. Atkinson, Maria E. Orlowska, Patrick Valduriez, Stanley B. Zdonik, and Michael L. Brodie, editors, *VLDB*, pages 223–234. Morgan Kaufmann, 1999. ISBN 1-55860-615-7.

[43] Jian Pei, Jiawei Han, and Wei Wang. Constraint-based sequential pattern mining: the pattern-growth methods. *Journal of Intelligent Information Systems*, 28(2):133–160, 2007.

[44] Jiong Yang, Wei Wang, Philip S. Yu, and Jiawei Han. Mining long sequential patterns in a noisy environment. In Michael J. Franklin, Bongki Moon, and Anastassia Ailamaki, editors, *SIGMOD Conference*, pages 406–417. ACM, 2002. ISBN 1-58113-497-5.

[45] Xingzhi Sun, Maria E. Orlowska, and Xue Li. Introducing uncertainty into pattern discovery in temporal event sequences. In *ICDM*, pages 299–306. IEEE Computer Society, 2003. ISBN 0-7695-1978-4.

[46] Zhou Zhao, Da Yan, and Wilfred Ng. Mining probabilistically frequent sequential patterns in uncertain databases. In Elke A. Rundensteiner, Volker Markl, Ioana Manolescu, Sihem Amer-Yahia, Felix Naumann, and Ismail Ari, editors, *EDBT*, pages 74–85. ACM, 2012. ISBN 978-1-4503-0790-1.

[47] Muhammad Muzammal and Rajeev Raman. Uncertainty in sequential pattern mining. In Lachlan M. MacKinnon, editor, *BNCOD*, volume 6121 of *Lecture*

Notes in Computer Science, pages 147–150. Springer, 2010. ISBN 978-3-642-25703-2.

[48] Muhammad Muzammal and Rajeev Raman. On probabilistic models for uncertain sequential pattern mining. In Longbing Cao, Yong Feng, and Jiang Zhong, editors, *ADMA (1)*, volume 6440 of *Lecture Notes in Computer Science*, pages 60–72. Springer, 2010. ISBN 978-3-642-17315-8.

[49] Muhammad Muzammal and Rajeev Raman. Mining sequential patterns from probabilistic databases. In Joshua Zhexue Huang, Longbing Cao, and Jaideep Srivastava, editors, *PAKDD (2)*, volume 6635 of *Lecture Notes in Computer Science*, pages 210–221. Springer, 2011. ISBN 978-3-642-20846-1.

[50] Muhammad Muzammal. Mining sequential patterns from probabilistic databases by pattern-growth. In Alvaro A. A. Fernandes, Alasdair J. G. Gray, and Khalid Belhajjame, editors, *BNCOD*, volume 7051 of *Lecture Notes in Computer Science*, pages 118–127. Springer, 2011. ISBN 978-3-642-24576-3.

[51] Yanchang Zhao, Huaifeng Zhang, Shanshan Wu, Jian Pei, Longbing Cao, Chengqi Zhang, and Hans Bohlscheid. Debt detection in social security by sequence classification using both positive and negative patterns. In Wray L. Buntine, Marko Grobelnik, Dunja Mladenic, and John Shawe-Taylor, editors, *ECML/PKDD (2)*, volume 5782 of *Lecture Notes in Computer Science*, pages 648–663. Springer, 2009. ISBN 978-3-642-04173-0.

[52] Ke Wang, Yabo Xu, and Jeffrey Xu Yu. Scalable sequential pattern mining for biological sequences. In David A. Grossman, Luis Gravano, ChengXiang Zhai, Otthein Herzog, and David A. Evans, editors, *CIKM*, pages 178–187. ACM, 2004. ISBN 1-58113-874-1.

[53] Miao Wang, Xuequn Shang, and Zhanhuai Li. Sequential pattern mining for protein function prediction. In Changjie Tang, Charles X. Ling, Xiaofang Zhou,

Nick Cercone, and Xue Li, editors, *ADMA*, volume 5139 of *Lecture Notes in Computer Science*, pages 652–658. Springer, 2008. ISBN 978-3-540-88191-9.

[54] Takashi Ishio, Hironori Date, Tatsuya Miyake, and Katsuro Inoue. Mining coding patterns to detect crosscutting concerns in Java programs. In Ahmed E. Hassan, Andy Zaidman, and Massimiliano Di Penta, editors, *WCRE*, pages 123–132. IEEE, 2008.

[55] Roger Cavallo and Michael Pittarelli. The theory of probabilistic databases. In Peter M. Stocker, William Kent, and Peter Hammersley, editors, *VLDB*, pages 71–81. Morgan Kaufmann, 1987. ISBN 0-934613-46-X.

[56] Nilesh N. Dalvi and Dan Suciu. Efficient query evaluation on probabilistic databases. In Mario A. Nascimento, M. Tamer Özsu, Donald Kossmann, Renée J. Miller, José A. Blakeley, and K. Bernhard Schiefer, editors, *VLDB*, pages 864–875. Morgan Kaufmann, 2004. ISBN 0-12-088469-0.

[57] Jihad Boulos, Nilesh N. Dalvi, Bhushan Mandhani, Shobhit Mathur, Christopher Ré, and Dan Suciu. Mystiq: a system for finding more answers by using probabilities. In Özcan [81], pages 891–893. ISBN 1-59593-060-4.

[58] Sumit Sarkar and Debabrata Dey. Relational models and algebra for uncertain data. In Charu C. Aggarwal, editor, *Managing and Mining Uncertain Data*, chapter 3. Springer, 2009.

[59] Nilesh N. Dalvi and Dan Suciu. The dichotomy of conjunctive queries on probabilistic structures. In Leonid Libkin, editor, *PODS*, pages 293–302. ACM, 2007. ISBN 978-1-59593-685-1.

[60] Mohamed A. Soliman, Ihab F. Ilyas, and Kevin Chen-Chuan Chang. Top-k query processing in uncertain databases. In Rada Chirkova, Asuman Dogac,

M. Tamer Özsu, and Timos K. Sellis, editors, *ICDE*, pages 896–905. IEEE, 2007.

[61] Ming Hua, Jian Pei, Wenjie Zhang, and Xuemin Lin. Ranking queries on uncertain data: a probabilistic threshold approach. In Wang [79], pages 673–686. ISBN 978-1-60558-102-6.

[62] Xi Zhang and Jan Chomicki. Semantics and evaluation of top-k queries in probabilistic databases. *Distributed and Parallel Databases*, 26(1):67–126, 2009.

[63] Chun Kit Chui and Ben Kao. A decremental approach for mining frequent itemsets from uncertain data. In Washio et al. [80], pages 64–75. ISBN 978-3-540-68124-3.

[64] Jiawei Han, Jian Pei, Yiwen Yin, and Runying Mao. Mining frequent patterns without candidate generation: A frequent-pattern tree approach. *Data Mining and Knowledge Discovery*, 8(1):53–87, 2004.

[65] Qin Zhang, Feifei Li, and Ke Yi. Finding frequent items in probabilistic data. In Wang [79], pages 819–832. ISBN 978-1-60558-102-6.

[66] Liang Wang, Reynold Cheng, Sau Dan Lee, and David Wai-Lok Cheung. Accelerating probabilistic frequent itemset mining: a model-based approach. In Jimmy Huang, Nick Koudas, Gareth J. F. Jones, Xindong Wu, Kevyn Collins-Thompson, and Aijun An, editors, *CIKM*, pages 429–438. ACM, 2010. ISBN 978-1-4503-0099-5.

[67] Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schtze. *Introduction to Information Retrieval*. Cambridge University Press, New York, NY, USA, 2008. ISBN 0521865719, 9780521865715.

[68] Wikipedia. http://en.wikipedia.org/wiki/anpr — Wikipedia, the free encyclopedia, 2010. URL `http://en.wikipedia.org/wiki/ANPR`. [Online; accessed 30-April-2012].

[69] Liam Keilthy. ANPR System performance. In Marko Ruh and Gerhard Trost-Heutmekers, editors, *Parking Trend International*. European Parking Association, June 2008.

[70] Richard M. Karp. Reducibility among combinatorial problems. In Raymond E. Miller and James W. Thatcher, editors, *Complexity of Computer Computations*, The IBM Research Symposia Series, pages 85–103. Plenum Press, New York, 1972. ISBN 0-306-30707-3.

[71] Leslie G. Valiant. The complexity of computing the permanent. *Theoretical Computer Science*, 8:189–201, 1979.

[72] Ron Kohavi, Carla Brodley, Brian Frasca, Llew Mason, and Zijian Zheng. KDD-Cup 2000 organizers' report: Peeling the onion. *SIGKDD Explorations*, 2(2): 86–98, 2000.

[73] Hye-Chung Kum, Joong Hyuk Chang, and Wei Wang. Benchmarking the effectiveness of sequential pattern mining methods. *Data and Knowledge Engineering*, 60(1):30–50, 2007.

[74] Li Wan. Discovering probabilistic sequential pattern in uncertain sequence database. In Gang Shen and Xiong Huang, editors, *Advanced Research on Computer Science and Information Engineering*, volume 153 of *Communications in Computer and Information Science*, pages 125–131. Springer Berlin Heidelberg, 2011. ISBN 978-3-642-21411-0.

[75] Manish Gupta, Jing Gao, Yizhou Sun, and Jiawei Han. Community trend outlier detection using soft temporal pattern mining. In *ECML/PKDD*, 2012. To Appear.

[76] Christos H. Papadimitriou. *Computational complexity*. Addison-Wesley, 1994. ISBN 978-0-201-53082-7.

[77] John F. Elder, Françoise Fogelman-Soulié, Peter A. Flach, and Mohammed Javeed Zaki, editors. *Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Paris, France, June 28 - July 1, 2009*, 2009. ACM. ISBN 978-1-60558-495-9.

[78] *Proceedings of the 24th International Conference on Data Engineering, ICDE 2008, April 7-12, 2008, Cancún, México*, 2008. IEEE.

[79] Jason Tsong-Li Wang, editor. *Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2008, Vancouver, BC, Canada, June 10-12, 2008*, 2008. ACM. ISBN 978-1-60558-102-6.

[80] Takashi Washio, Einoshin Suzuki, Kai Ming Ting, and Akihiro Inokuchi, editors. *Advances in Knowledge Discovery and Data Mining, 12th Pacific-Asia Conference, PAKDD 2008, Osaka, Japan, May 20-23, 2008 Proceedings*, volume 5012 of *Lecture Notes in Computer Science*, 2008. Springer. ISBN 978-3-540-68124-3.

[81] Fatma Özcan, editor. *Proceedings of the ACM SIGMOD International Conference on Management of Data, Baltimore, Maryland, USA, June 14-16, 2005*, 2005. ACM. ISBN 1-59593-060-4.

[82] *Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, July 23-26, 2002, Edmonton, Alberta, Canada*, 2002. ACM. ISBN 1-58113-567-X.