Characterizing Uncertain Data using Compression

Francesco Bonchi
Yahoo! Research
Barcelona, Spain
bonchi@yahoo-inc.com

Matthijs van Leeuwen
Universiteit Utrecht
Utrecht, The Netherlands
mleeuwen@cs.uu.nl

Antti Ukkonen
Yahoo! Research
Barcelona, Spain
aukkonen@yahoo-inc.com

Abstract

Motivated by sensor networks, mobility data, biology and life sciences, the area of mining uncertain data has recently received a great deal of attention. While various papers have focused on efficiently mining frequent patterns from uncertain data, the problem of discovering a small set of interesting patterns that provide an accurate and condensed description of a probabilistic database is still unexplored.

In this paper we study the problem of discovering characteristic patterns in uncertain data through information theoretic lenses. Adopting the *possible worlds* interpretation of probabilistic data and a compression scheme based on the MDL principle, we formalize the problem of mining patterns that compress the database well in expectation. Despite its huge search space, we show that this problem can be accurately approximated.

In particular, we devise a sequence of three methods where each new method improves the memory requirements orders of magnitudes compared to its predecessor, while giving up only a little in terms of approximation accuracy. We empirically compare our methods on both synthetic data and real data from life science. Results show that from a probabilistic matrix with more than one million rows and columns, we can extract a small set of meaningful patterns that accurately characterize the data distribution of any probable world.

1 Introduction

In pattern mining it is generally assumed that data is certain. That is, a transaction in a database consists of a fixed set of items, which is realistic for the market basket domain: a certain product is either present or not in a customer's basket, with no possibility of uncertainty or noise in the data. However, uncertainty is inherent to data in many other domains: noisy measurements, incompleteness, data loss, and privacy preserving perturbation processes are only some of the possible causes of uncertainty. To this end, several pattern mining methods for uncertain data have recently been proposed [10, 9, 5, 3, 7]. Most of them assume statistical independence of the items in all transactions and adopt the *possible worlds* interpretation of probabilistic data [33, 13]. In this setting, a probabilistic transactional dataset $\mathcal{P} \in [0,1]^{n \times m}$ is given, where $\mathcal{P}_{i,j}$ indicates the probability of the jth item being present in the ith transaction, and n resp. m are the total number of transactions resp. items. In the real world $\mathcal{P}_{i,j}$ would be either 0 or 1, but in the probabilistic setting we have no certainty about this. Instead, we have $2^{n \times m}$ possible worlds D sampled from $\mathcal{P}$ (denoted $D \sqsubseteq \mathcal{P}$). Given this interpretation, the frequency of a pattern X can be naturally defined as the *expected support*:

$$\mathrm{E}[\mathrm{support}(X)] = \sum_{D \sqsubseteq \mathcal{P}} \Pr(D) \times \mathrm{support}_D(X), \tag{1.1}$$

where $\mathrm{support}_D(X)$ is defined as the number of transactions $t \in D$ for which $X \subseteq t$.

However, mining frequent patterns from probabilistic data suffers from the same problems as 'regular' frequent pattern mining. The main problem is the well-known *pattern explosion*: for low minimum supports, huge amounts of patterns are found. Moreover, many redundant patterns are usually discovered, since patterns are only judged by their individual merits.

In this paper we study the problem of discovering a small set of 'interesting' patterns that provide a good characterization of an uncertain database. In particular we use the Minimum Description Length (MDL) principle, which has been shown to be a useful approach in many data mining tasks [15]. In short, the idea is to use frequent patterns to construct a model that describes the data in the best possible way. Pioneered in [27] for the problem of pattern selection, the same approach has been shown useful for classification [23], characterizing differences between data sets [31], and change detection in data streams [22].

Given a probabilistic database as the only input, our goal is to describe the 'real world' using a compression model. In particular, the kind of model we are interested in is the so-called *code table*, i.e. a small set of patterns, each with its own associated code.

A database can be compressed by replacing occurrences of code table patterns with their respective codes, obtaining lossless compression. The better the compression is, the better is the model, and the more descriptive are the patterns for the data [27]. Since we do not know which of the many possible worlds is the real one, it is crucial to induce a description (a code table) that is accurate for all possible worlds, proportionally to their probabilities. In other terms, our problem is to discover *patterns that compress well in expectation*.

Contributions and Roadmap - To the best of our knowledge, this is the first work taking an MDL perspective on finding condensed and accurate descriptions of uncertain data. The main contributions are introducing, characterizing and solving a novel pattern mining problem.

In the next section we discuss related work. In Section 3 we provide the needed background on MDL for transaction data and define our models: code tables. In the subsequent section we formalize the problem of mining the *expected optimal code table*. In Section 5 we prove that we can solve our problem by materializing the $2^m \times m$ binary matrix containing all possible transactions, weighting each such transaction with the number of times that we *expect* to see it, and then applying a standard deterministic method to this weighted matrix. The proposed technique can be generalized to compute other measures on uncertain data, for instance the expected support of itemsets. Obviously, due to the 2^m term, it can only be applied when m is small. When this is the case, this method can be used for an interesting comparison as it allows to adopt the algorithms developed for the deterministic setting.

For larger m we have to rely on sampling. We prove that by sampling a set of possible worlds and concatenating them into a single database, the expected compressed size of any possible world can be accurately estimated with a sample mean based estimator. In Section 6, we derive statistical bounds on the approximation accuracy achievable by sampling k possible worlds.

However, concatenating k databases to obtain a single database can be costly in terms of memory requirements. Moreover, we would like to avoid having k as a parameter. These considerations lead to the third method, dubbed *SMUC* and introduced in Section 7. *SMUC* is also based on sampling, but instead of concatenating k samples at once, it considers samples one by one, incrementally updating the code table, and halting the process when the code table stops changing. Our three methods are empirically compared in Section 8. While previous work on mining uncertain data almost always uses synthetic data, our experiments are conducted on both synthetic and real data from life sciences. The results show that from a probabilistic matrix with more than one million of rows and columns we can extract a small set of meaningful patterns that accurately capture the data distribution of probable worlds.

2 Related Work

Research in uncertain and probabilistic data management is abundant (see e.g. [2, 12] for recent surveys). Recently, also the data mining community has started dealing with uncertain data, tackling the problems of clustering [11, 4, 20], classification [25, 30] and frequent pattern mining, but finding small and informative pattern-based descriptions of uncertain databases has not been addressed before.

Particularly relevant for our work is the research in frequent pattern mining from uncertain data. To the best of our knowledge, the first paper tackling this problem is by Chui et al. [10]. Their approach is based on the possible worlds interpretation of probabilistic data, originally introduced by Zimányi and Pirotte [33], that we adopt for this paper as well. The proposed algorithm mines all patterns that have at least a given expected support (as defined in the Introduction). Bernecker et al. [5] propose efficient algorithms to mine Probabilistic Frequent Itemsets, itemsets that are frequent with at least a given probability. Aggarwal et al. [3] provide a comparison of different algorithmic techniques for mining frequent itemsets in the context of uncertain data. Sun et al. [28] present algorithms for mining association rules in uncertain data. Recently, Calders et al. [7] showed that sampling is a powerful tool when estimating supports of itemsets. We use similar techniques to argue that our MDL models can be accurately estimated by using a sampling based approach.

In the context of certain data, various methods for pattern set selection have been proposed. Knobbe and Ho [21] and Bringmann and Zimmermann [6] introduced techniques for selecting informative patterns. In their work, patterns sets have to be as small and informative as possible, and are used in tasks like classification. Another method that provides a lossy description of the data is Summarization [8], which uses compression to identify a group of itemsets such that each transaction is summarized by one itemset with as little loss of information as possible. Wang and Karypis [32] find summary sets, sets of itemsets that contain the largest frequent itemset covering each transaction.

A final family of methods provides lossless data descriptions using compression. Siebes et al. [27] pioneered this direction with Krimp, which uses MDL to find code tables on deterministic data. We will use this as part of our algorithms. Two approaches inspired by Krimp are Pack [29] (decision trees that transmit attributes) and LESS [18] (using low-entropy sets).

3 Background: MDL for Transaction Data

We adopt the standard notation from frequent itemset mining literature: a transactional dataset D over an alphabet of items $\mathcal{I}$ is a bag of sets (or transactions) $t \subseteq \mathcal{I}$. Alternatively we can regard a transactional dataset D as a binary matrix with $m = |\mathcal{I}|$ columns, and n rows $t_1, \ldots, t_n$ corresponding to the transactions.

MDL [17] is a practical version of Kolmogorov Complexity [15]. Both embrace the slogan *Induction by Compression*. For MDL, this principle can be roughly described as follows. Given a transactional dataset D and a set of models $\mathcal{H}$, the best model $H \in \mathcal{H}$ is the one that minimizes $L(H) + L(D|H)$, where $L(H)$ is the length, in bits, of the description of H, and $L(D|H)$ is the length, in bits, of the description of the data when encoded with H. In order to use this principle we need to define our models and how to encode the data with such a model. We use code tables as our models. The remainder of this section is mostly taken from [27] and presented here for the convenience of the reader.

Definition 1. (Code Table [27]) *Let $\mathcal{I}$ be a set of items and $\mathcal{C}$ a set of code words from a prefix code. A code table CT over $\mathcal{I}$ and $\mathcal{C}$ is a two column table. The first column contains itemsets over $\mathcal{I}$, and is laid out in coding order, i.e. descending on 1) itemset length, 2) support and 3) lexicographically. It always contains at least the singleton itemsets. The second column contains elements from $\mathcal{C}$, one for each itemset in the first column, such that each element of $\mathcal{C}$ occurs at most once. An itemset $X \in CT$ iff X occurs in the first column of CT, similarly for a code $C \in \mathcal{C}$. For $X \in CT$, $\text{code}_{CT}(X)$ denotes its code, i.e. the corresponding element in the second column. Finally, $|CT|$ denotes the number of itemsets in CT.*

To encode a transaction database D over $\mathcal{I}$ with code table CT, we use the Cover algorithm from [27] given in Algorithm 1. Its parameters are a code table CT and a transaction t, the result is a set of elements of CT that cover t. Note that Cover is a well-defined function on any code table and any transaction t, since CT contains at least the singletons. The cover of a transaction t given CT, denoted $\text{Cover}(CT, t)$, is the set of itemsets returned by the Cover algorithm with t and CT as the input. To encode database D, we simply replace each transaction by the codes of the itemsets in its cover: $t \to \{\text{code}_{CT}(X) \mid X \in \text{Cover}(CT, t)\}$. Note that since $\mathcal{C}$ is a prefix code, we can decode an encoded database uniquely.

Since MDL is concerned with the best compression, the codes in CT should be chosen such that the most often used itemsets have the shortest codes. Fortunately, there is a nice correspondence between codes and probability distributions (see, e.g. [24]): given a probability distribution $\mathbb{P}$ on some finite set A, there exists a code on A such that the length of the code for $a \in A$, denoted by $L(a)$, is given by $L(a) = -\log(\mathbb{P}(a))$. Moreover, this code is optimal in the sense that it gives the smallest expected encoded size for inputs drawn according to $\mathbb{P}$.

Algorithm 1 The Cover Algorithm

Cover(CT, t) :
$Res \leftarrow$ first $X \in CT$ in coding order for which $X \subseteq t$
if $t \setminus Res \neq \emptyset$ **then** $Res \leftarrow Res \cup$ **Cover**$(CT, t \setminus Res)$
return Res

Therefore, to produce codes for the itemsets, we construct a probability distribution having the property that commonly used itemsets have high probabilities. This distribution should depend on the data D, as well as CT, since we want to find a CT that results in the shortest coding length given data D. A natural choice is to use the usage counts of the itemsets, as commonly used itemsets should be assigned shorter codes. The usage of $X \in CT$ is the number of transactions in D that are covered by X:

$$\text{usage}_D(X \mid CT) = \sum_{t \in D} \mathbb{I}\{X \in \text{Cover}(CT, t)\}, \tag{3.2}$$

where $\mathbb{I}\{\cdot\}$ is the indicator function. To induce a probability distribution over the itemsets in the code table we consider the relative usages:

$$\mathbb{P}_D(X \mid CT) = \frac{\text{usage}_D(X \mid CT)}{\sum_{Y \in CT} \text{usage}_D(Y \mid CT)}. \tag{3.3}$$

A code is optimal for D iff $L(\text{code}_{CT}(X)) = -\log(\mathbb{P}_D(X \mid CT))$, and CT is code-optimal for D if all its codes $C \in CT$ are optimal for D. From now on, we assume that code tables are code-optimal.

For any database D and code table CT, we can now compute $L(D|CT)$. The encoded size of a transaction, denoted by $L(t|CT)$, is the sum of the sizes of the codes of the itemsets in its cover. That is,

$$L(t \mid CT) = \sum_{X \in \text{Cover}(CT, t)} L(\text{code}_{CT}(X)). \tag{3.4}$$

The encoded size of the transactional dataset D, denoted by $L(D|CT)$, is the sum of the sizes of its transactions, i.e. $L(D|CT) = \sum_{t \in D} L(t|CT)$. The remaining problem is to determine the size of a code table. For the second column this is clear as we know the size of each of the codes. For encoding the first column, we use the simplest code table, i.e. the code table that contains only the singleton elements and is code-optimal for D. This code table is called the *standard code table* and is denoted by ST. With this choice the size of CT, denoted by $L(CT)$, is given by $L(CT) = \sum_{X \in CT} L(\text{code}_{ST}(X)) + L(\text{code}_{CT}(X))$.

With these results we know the total size of our encoded database. It is the sum of the size of the encoded database plus the size of the code table. The total size of the encoded database, denoted by $L(D, CT)$, is given by $L(D, CT) = L(D|CT) + L(CT)$.

Clearly, two different code tables may yield different encoded sizes. The lower the total encoded size is, the better the code table captures the structure of the database. An *optimal code table* is one that minimizes the total size.

4 Problem Definition

We are given $\mathcal{P} \in [0,1]^{n \times m}$, i.e. a probabilistic transactional dataset or a matrix of probabilities, where $\mathcal{P}_{i,j}$ indicates the probability of the jth item being present in the ith transaction. We adopt the *possible worlds* based model of uncertainty [33]. A possible world is a binary matrix $D \in \{0,1\}^{n \times m}$ (or a transactional dataset) sampled from $\mathcal{P}$ according to the probabilities $\mathcal{P}_{i,j}$, meaning that $\Pr(D_{i,j} = 1) = \mathcal{P}_{i,j}$. There are $2^{n \times m}$ possible worlds D sampled from $\mathcal{P}$ (denoted $D \sqsubseteq \mathcal{P}$) and the probability of an individual world D is given by

$$\Pr(D) = \prod_i \prod_j \big(D_{i,j}\mathcal{P}_{i,j} + (1 - D_{i,j})(1 - \mathcal{P}_{i,j})\big).$$

Note that this implies that all transactions and items are assumed to be independent.

The probabilistic database $\mathcal{P}$ can be interpreted as our belief about the true state of the world. That is, reality is one of the possible worlds, a binary matrix, but our observations of it contain uncertainty, reflected by the probabilities in $\mathcal{P}$. Since we cannot be sure about the true state of the world, we want to mine a model that works well in any probable world; a possible world induced by $\mathcal{P}$ with relatively high $\Pr(D)$. In other terms, we want to mine a single code table CT^* that is good *in expectation*.

Abstractly, given a probabilistic transactional dataset $\mathcal{P}$ the problem is to find the code table CT^* with *minimum expected coding length*. The first step towards a proper formalization of this problem is the definition of the optimal codes. Recall that our definition of $\mathbb{P}_D(X \mid CT)$ (Eq. 3.3) is based on the usages of the itemsets. Itemsets with higher usages should get shorter codes than itemsets with lower usages. This idea carries over to probabilistic databases in a straightforward manner if we consider the *expected usage* of an itemset. Denote by $\mathrm{E}_{D \sqsubseteq \mathcal{P}}[f(D)]$ the expected value of $f(D)$, where f is some function and D a database sampled from the distribution specified by $\mathcal{P}$. For simplicity we will often use E to denote $\mathrm{E}_{D \sqsubseteq \mathcal{P}}$, as the expectations are always taken over the probability distribution induced by $\mathcal{P}$. The expected usage of the itemset X when covering a probabilistic dataset $\mathcal{P}$ using the code table CT is thus given by

$$\mathrm{E}_{D \sqsubseteq \mathcal{P}}[\mathrm{usage}_D(X \mid CT)] = \sum_{D \sqsubseteq \mathcal{P}} \Pr(D) \times \mathrm{usage}_D(X \mid CT),$$

where D is a possible world, $\Pr(D)$ is the probability of observing this world given $\mathcal{P}$, and $\mathrm{usage}_D(X \mid CT)$ is the usage of X when covering D using CT. Analogous to the deterministic case, we can again construct a probability distribution on the itemsets in the code table CT by normalizing the expected usages:

$$\mathbb{P}_{\mathcal{P}}(X \mid CT) = \frac{\mathrm{E}[\mathrm{usage}_D(X \mid CT)]}{\sum_{Y \in CT} \mathrm{E}[\mathrm{usage}_D(Y \mid CT)]}. \tag{4.5}$$

The optimal length of the code for itemset X in a code table CT is now defined as $L(\mathrm{code}_{CT}(X)) = -\log(\mathbb{P}_{\mathcal{P}}(X|CT))$. Clearly itemsets that are *expected* to be used often are assigned short codes. The set of code-optimal code tables, denoted $\mathcal{CT}$, consists of code tables that use these code lengths. We are now ready to formally define the problem tackled in this paper.

Problem 1. (Expected Optimal Code Table) *Given a probabilistic dataset $\mathcal{P} \in [0,1]^{n \times m}$, find the code table $CT^* \in \mathcal{CT}$ such that:*

$$CT^* = \operatorname*{argmin}_{CT \in \mathcal{CT}} \mathrm{E}_{D \sqsubseteq \mathcal{P}}[L(D, CT)].$$

By linearity of expectation and the fact that the cost of the code table does not depend on D, we have $\mathrm{E}[L(D, CT)] = \mathrm{E}[L(D|CT)] + L(CT)$. Hence the challenging part is the computation of $\mathrm{E}[L(D|CT)]$: a naïve algorithm would enumerate all possible worlds of $\mathcal{P}$, which is obviously untractable. In Section 5 we discuss an exact method for computing $\mathrm{E}[L(D|CT)]$ that is tractable in cases where the number of items is relatively small. In Section 6 we argue that a good estimate of $\mathrm{E}[L(D|CT)]$ can be obtained by sampling.

4.1 Search space size. The problem of finding the code table that minimizes the total encoded size averaged over all possible worlds has a huge search space. In [27][Lemma 2.4] it is proven that for an alphabet of items $\mathcal{I}$, the number of (code-optimal) code tables is

$$\sum_{j=0}^{2^{|\mathcal{I}|} - |\mathcal{I}| - 1} \binom{2^{|\mathcal{I}|} - |\mathcal{I}| - 1}{j} \times (j + |\mathcal{I}|)!$$

Given a probabilistic dataset $\mathcal{P} \in [0,1]^{n \times m}$, where $m = |\mathcal{I}|$, an exhaustive algorithm would need to compute $L(D, CT)$ for $2^{n \times m}$ possible worlds D and all the possible code tables $CT \in \mathcal{CT}$: this is way too much!

As an example, for a rather small probabilistic matrix with 100 rows and 6 columns, we would have to compute $L(D, CT)$ for a number of times equal to

$$4.90 \times 10^{87} \times 2^{600} \approx 2 \times 10^{268}.$$

In the remainder of this paper we develop methods to accurately approximate the expected optimal code table for probabilistic transactional datasets with up to *millions of rows and columns*.

5 Exact computation of $\mathrm{E}[L(D|CT)]$

In this section we present a method that can be applied when the number of items $m = |\mathcal{I}|$ is small. In this case we can reduce our problem to its deterministic version by materializing a transactional database with the transactions (rows) weighted by their expected counts in $\mathcal{P}$. Given a transactional database D and a transaction t, let $\mathrm{count}_D(t)$ denote the number of times t appears in D. The following proposition states that we can compute the expected count of a transaction t without summing over all possible worlds.

Proposition 5.1. *For any probabilistic database $\mathcal{P}$, and every m-dimensional 0-1 vector t we have*

$$\mathrm{E}[\mathrm{count}_D(t)] = \sum_{i=1}^{n} \prod_{j=1}^{m} \big(t_j \mathcal{P}_{ij} + (1 - t_j)(1 - \mathcal{P}_{ij})\big).$$

Proof. Let D_i denote the ith row of D, and let $\mathbb{I}\{\cdot\}$ be the indicator function. By linearity of expectation we have:

$$\mathrm{E}[\mathrm{count}_D(t)] = \mathrm{E}[\sum_{i=1}^{n} \mathbb{I}\{D_i = t\}] = \sum_{i=1}^{n} \mathrm{E}[\mathbb{I}\{D_i = t\}] =$$

$$= \sum_{i=1}^{n} \Pr(D_i = t) = \sum_{i=1}^{n} \prod_{j=1}^{m} \big(t_j \mathcal{P}_{ij} + (1 - t_j)(1 - \mathcal{P}_{ij})\big).$$

□

Our second step is to show that the expected counts can be used also to compute expectations of certain other functions, including $L(D \mid CT)$. Let M be the $2^m \times m$ binary matrix where the rows correspond to every possible (nonzero) m-dimensional binary vector t. Moreover, let $M(\mathcal{P})$ denote M where each row t has $\mathrm{E}[\mathrm{count}_D(t)]$ attached. It turns out that given $M(\mathcal{P})$ it is possible to compute $\mathrm{E}[L(D \mid CT)]$ exactly, without summing over all possible worlds.

Actually, we give a more general result that applies to any function f on transactional databases D that decomposes into a sum over the transactions in D. Note that the expected coding length, but also e.g. the expected support of an itemset X are special cases.

Lemma 5.1. *Let $\mathcal{P}$ be a probabilistic dataset, define the matrix M as above, and let f be a function on transactional databases st. $f(D) = \sum_{t \in D} g(t)$, where g is a function that only depends on t. We have*

$$\mathrm{E}[f(D)] = \sum_{t \in M} g(t)\, \mathrm{E}[\mathrm{count}_D(t)].$$

Proof. The proof is a simple matter of rearranging the sums:

$$\begin{aligned}
\mathrm{E}[f(D)] &= \sum_{D \sqsubseteq \mathcal{P}} \Pr(D) f(D) = \sum_{D \sqsubseteq \mathcal{P}} \Pr(D) \sum_{t \in D} g(t) \\
&= \sum_{D \sqsubseteq \mathcal{P}} \Pr(D) \sum_{t \in M} g(t)\, \mathrm{count}_D(t) \\
&= \sum_{t \in M} g(t) \sum_{D \sqsubseteq \mathcal{P}} \Pr(D)\, \mathrm{count}_D(t) \\
&= \sum_{t \in M} g(t)\, \mathrm{E}[\mathrm{count}_D(t)].
\end{aligned}$$

□

Lemma 5.1 is fairly general, as many interesting functions can be expressed as a sum over all transactions in a transactional dataset. As an example, consider the *expected support* of an itemset X in a probabilistic transactional dataset $\mathcal{P}$ (Eq. 1.1). This is clearly of the required form if we let $g(t) = \mathbb{I}\{X \subseteq t\}$, as $\mathit{support}(X) = \sum_{t \in D} \mathbb{I}\{X \subseteq t\}$. Thus from Lemma 5.1 the next corollary follows.

Corollary 5.1. *Given a probabilistic database $\mathcal{P}$, the* expected support *of an itemset X can be computed on $M(\mathcal{P})$ as:*

$$\mathrm{E}[\mathrm{support}_D(X)] = \sum_{t \in M} \mathbb{I}\{X \subseteq t\}\, \mathrm{E}[\mathrm{count}_D(t)].$$

More importantly, also the coding length $L(D \mid CT)$ is clearly within the scope of Lemma 5.1. We have $L(D \mid CT) = \sum_{t \in D} L(t \mid CT)$, and letting $g(t) = L(t \mid CT)$ directly results in the following:

Corollary 5.2. *Given a probabilistic database $\mathcal{P}$ and a code table CT, it holds that*

$$\mathrm{E}[L(D \mid CT)] = \sum_{t \in M} L(t \mid CT)\, \mathrm{E}[\mathrm{count}_D(t)].$$

Next we have to show that also the expected usages of the itemsets $X \in CT$ can be computed directly on $M(\mathcal{P})$. Recall that

$$\mathrm{usage}_D(X \mid CT) = \sum_{t \in D} \mathbb{I}\{X \in \mathrm{Cover}(CT, t)\}.$$

Observe that the Cover function only depends on CT and t. In particular, it does *not* depend on D. Therefore we can let $g(t) = \mathbb{I}\{X \in \text{Cover}(CT, t)\}$, and obtain the final result of this section.

COROLLARY 5.3. *Given a probabilistic database $\mathcal{P}$ and a code table CT,*

$$\text{E}[\text{usage}_D(X \mid CT)] = \sum_{t \in M} \mathbb{I}\{X \in \text{Cover}(CT, t)\} \, \text{E}[\text{count}_D(t)].$$

In conclusion, minimizing the expected coding length of D is equivalent to finding the optimal code table on the deterministic matrix $M(\mathcal{P})$, keeping in count the weights of its rows $\text{E}[\text{count}_D(t)]$. Therefore a naïve method to solve Problem 1 is to materialize the $2^m \times m$ binary matrix where the rows correspond to every possible (nonzero) m-dimensional binary vector t, compute the expected count in $\mathcal{P}$ of every such vector t, and apply the standard algorithm developed in [27] for the deterministic setting. We dub this method *MED* (for "Materialize Expectation in Data").

Computing $\text{E}[\text{count}_D(t)]$ for a given t can be carried out in $O(nm)$ time, and hence the complexity of materializing the matrix $M(\mathcal{P})$ is of order $O(2^{m+1}n)$. When the number of items m is small enough, this method provides us a baseline to which other methods can be compared, as code tables are mined by the algorithms developed for the deterministic setting, applied to a sort of "expected database". Clearly, this approach becomes quickly infeasible as m grows. For databases with a large number of items we have to rely on sampling.

6 Approximating $\text{E}[L(D|CT)]$ by sampling

In the basic sampling approach we sample a set $D_1, \ldots D_k$ of k possible worlds from $\mathcal{P}$ according to their existential probabilities, concatenate them into a single database, denoted by D^k, and then find a code table that gives a good description of D^k. Sampling a database D_i requires to flip a coin of bias $\mathcal{P}_{i,j}$ for each $\mathcal{P}_{i,j} \in \mathcal{P}$ (i.e. $n \times m$ coin flips).

In this section we argue that this seemingly trivial approach gives a good approximation to Problem 1. In particular we next provide statistical bounds on the accuracy of the approximation of $\text{E}[L(D|CT)]$ achievable by sampling.[1] To do this, first observe that the sample mean based estimator for $\text{E}[L(D|CT)$ is defined as

$$\hat{\text{E}}[L(D|CT)] = \frac{1}{k} \sum_{i=1}^{k} L(D_i|CT).$$

[1]Recall that our Problem 1 requires to find code tables that minimize $\text{E}[L(D, CT)] = \text{E}[L(D|CT)] + L(CT)$, where $L(CT)$ does not depend on the possible worlds.

Since $L(D|CT) = \sum_{t \in D} L(t|CT)$, we can write

$$\hat{\text{E}}[L(D|CT)] = \frac{1}{k} \sum_{i=1}^{k} \sum_{t \in D_i} L(t|CT) = \frac{1}{k} \sum_{t \in D^k} L(t|CT).$$

That is, if the code lengths $L(t|CT)$ that we compute from D^k are accurate (in a sense to be made precise below), then we obtain an accurate estimate of $\text{E}[L(D|CT)]$ by dividing the coding length of D^k by k.

Since $L(t|CT)$ is simply a sum of the code lengths of all itemsets that belong to $\text{Cover}(CT, t)$, it is obvious that for $L(t|CT)$ to be estimated accurately, the code length of every itemset $X \in CT$ must be estimated accurately. Recall that we have $L(\text{code}_{CT}(X)) = -\log(\mathbb{P}_{\mathcal{P}}(X))$, and note that $\mathbb{P}_{\mathcal{P}}(X)$ depends on both the expected usage of X and the expected usages of all itemsets in CT. Therefore, obtaining an accurate estimate of the code length essentially depends on accurately estimating $\text{E}[\text{usage}(X \mid CT)]$ for all $X \in CT$.

For ease of exposition, we let μ_X and $\hat{\mu}_X$ denote $\text{E}[\text{usage}(X \mid CT)]$ and its estimator $\hat{\text{E}}[\text{usage}(X \mid CT)]$, respectively. Furthermore, denote by Γ and $\hat{\Gamma}$ the sums $\sum_{Y \in CT} \mu_Y$ and $\sum_{Y \in CT} \hat{\mu}_Y$, respectively. Using this notation, we can rewrite Equation 4.5 as

$$\mathbb{P}_{\mathcal{P}}(X) = \frac{\mu_X}{\Gamma}.$$

In the following we first show that both μ_X and Γ can be estimated accurately. A corollary of these results is that the estimate of $\mathbb{P}_{\mathcal{P}}(X)$ is close to its expected value, and thereby the code lengths are close to their expected values.

We start by defining the estimators of μ_X and Γ. As above with $L(D|CT)$, we consider sample mean based estimators, and let

$$\hat{\mu}_X = \sum_{t \in D^k} \frac{\mathbb{I}\{X \in \text{Cover}(CT, t)\}}{k}, \tag{6.6}$$

$$\hat{\Gamma} = \sum_{Y \in CT} \sum_{t \in D^k} \frac{\mathbb{I}\{Y \in \text{Cover}(CT, t)\}}{k}. \tag{6.7}$$

PROPOSITION 6.1. *Let $\hat{\mu}_X$ and $\hat{\Gamma}_X$ be sample mean based estimators of μ_X and Γ_X, respectively, using k independent samples, $D_1, \ldots, D_k$ from $\mathcal{P}$. We have*

$$\Pr\left(|\mu_X - \hat{\mu}_X| \geq \epsilon n\right) \leq 2\exp(-2nk\epsilon^2),$$

and

$$\Pr\left(|\Gamma - \hat{\Gamma}| \geq \Gamma S\epsilon\right) \leq 2\exp(-2nk\epsilon^2),$$

where n is the number of transactions in $\mathcal{P}$, and S the maximum number of items in a transaction sampled from $\mathcal{P}$.

Proof. The result is based on Hoeffding's Inequality [19]. Observe that while the indicator variables in Equation 6.7 are not completely independent, the dependent sets of variables can be grouped together and represented as bounded, independent variables. □

With these we are ready to state the main result of this section: for itemsets X having a reasonably high expected usage, the estimator for $\mathbb{P}_{\mathcal{P}}(X)$ is concentrated around its expectation with high probability.

PROPOSITION 6.2. *Let $\hat{\mathbb{P}}_{\mathcal{P}}(X)$ be an estimator of $\mathbb{P}_{\mathcal{P}}(X)$ based on the estimated expected usages. For any itemset $X \in CT$ st. $\mu_X \geq n/c$, we have*

$$\Pr\left(\hat{\mathbb{P}}_{\mathcal{P}}(X) \in \left[(1-x)\mathbb{P}_{\mathcal{P}}(X), (1+x)\mathbb{P}_{\mathcal{P}}(X)\right]\right) \geq (1-p)^2,$$

where $x = \frac{S+c}{S+\epsilon^{-1}}$, and $p = 2\exp(-2nk\epsilon^2)$.

Proof. The result follows from proposition 6.1, and the assumption that $\mu_X \geq n/c$. □

For example, if $c = 10$, $\epsilon = 0.01$, and $S = 5$, we have $x \approx 0.14$, and the code length of X (in bits) is with high probability contained in the range

$$[L(\text{code}_{CT}(X)) - 0.22, L(\text{code}_{CT}(X)) + 0.19],$$

i.e. the estimated code length of X lies within an interval of approximately 0.4 bits in width centered at $L(\text{code}_{CT}(X))$.

7 Methods

We have shown that we can approach our problem either by *materializing* a sort of "expected database" $M(\mathcal{P})$ (Sec. 5), or by *sampling* possible worlds and concatenating them in a database D^k (Sec. 6). In both cases we transform our input probabilistic database into a deterministic one, and therefore we can apply a method developed for the deterministic setting (KRIMP). We next recall KRIMP and summarize the two methods discussed above: *MED* and *SMM*. Finally we introduce *SMUC*, a method that overcomes the limits of the previous methods by approximating the expected optimal code table by incremental construction.

KRIMP is a heuristic algorithm that approximates the optimal code table for a (deterministic) transaction database [27], and needs a set of candidate frequent itemsets as input. The candidate set is ordered first descending on support, second descending on itemset cardinality and third lexicographically. KRIMP starts with the code table containing only singleton itemsets (named *Standard Code Table*, ST). One by one, each pattern in the candidate set is added to the code table to see if it helps to improve total compression. If it does, it is kept in the code table, otherwise it is removed. After this decision, the next candidate is tested. If post-accept pruning is applied, each time an itemset is accepted in the code table, all other itemsets are tested to see whether they still contribute to compression. Itemsets that do not, are permanently removed.

***MED*: "Materialize Expectation in Data".** When m is modest, the Expected Optimal Code Table can be easily approximated by applying KRIMP to the deterministic matrix $M(\mathcal{P})$, containing all possible non-zero transactions. The only required modification to the standard KRIMP algorithm is that the expected count of each row must be taken into account as a weight. This method was already outlined in Section 5 and dubbed *MED*. Of the three methods, it is the most accurate, since the only approximation is in the heuristics of KRIMP. On the downside, it has the largest memory requirements and it is infeasible for large m.

***SMM*: "Sample, Merge and Mine".** In Section 6 it was shown that by sampling possible worlds and concatenating these, a good approximation of the Expected Optimal Code Table can be found. Hence, the method is simple: sample k worlds from the probabilistic dataset, concatenate (or 'merge') them and apply KRIMP to the resulting database D^k. In this case, the only required modification is the computation of the compressed size: $\mathrm{E}[L(D \mid CT)]$ is estimated with $L(D^k \mid CT)/k$. The number of samples k is an input parameter. We dub this method *SMM*.

***SMUC*: "Sample & Mine Until Convergence".** The merging approach of *SMM* has two major disadvantages. First, it needs the input parameter k. Second, for larger k and larger datasets, maintaining all sampled worlds might be prohibitive in terms of memory usage.

Therefore, we next propose a third and entirely novel method that approximates the expected optimal code table by incremental construction. As such, it is the least accurate of the three methods, but it requires the least memory, thus scaling to very large probabilistic datasets. It starts with an empty code table and iteratively improves it by considering sampled worlds one by one. It continues until the code table no longer changes. We dub this method *SMUC* ("Sample & Mine Until Convergence"). SMUC is detailed in pseudo code in Algorithm 2.

SMUC starts with computing the expected standard code table (denoted by $\mathrm{E}[ST]$ on line 1), containing all singletons $i \in \mathcal{I}$, where each singleton has an asso-

Algorithm 2 The *SMUC* Algorithm

Input: Probabilistic database $\mathcal{P}$, threshold *minsup*
Output: Approximation of the optimal code table CT^*

1: $CT^* \leftarrow \mathrm{E}[ST]$
2: $U \leftarrow \emptyset$
3: **while** CT^* changes **do**
4: $\quad D \leftarrow \text{SampleWorld}(\mathcal{P})$
5: $\quad \mathcal{C} \leftarrow \text{MineCandidates}(D,\ minsup)$
6: $\quad$ **for all** $C \in \mathcal{C}$ **do**
7: $\quad\quad CT \leftarrow CT^* \cup C$
8: $\quad\quad \text{PrePrune}(CT, U, D)$
9: $\quad\quad$ **if** $L(CT, U, D) < L(CT^*, U, D)$ **then**
10: $\quad\quad\quad CT^* \leftarrow CT$
11: $\quad\quad\quad \text{PostPrune}(CT^*, U, D)$
12: $\quad$ **for all** $X \in CT^*$ **do**
13: $\quad\quad U[X] \leftarrow U[X] + usage_D(X)$
14: **return** CT^*

ciated code of length implied by its expected support. (Note that this is simply the sum of all probabilities in the i_{th} column of $\mathcal{P}$.)

Next, the main loop is entered: while the code table changes, sample worlds and improve the code table (3-13). First, a possible world is sampled and frequent itemsets are mined from this world (4-5). Each of these itemsets is then added to the code table (7), using the same candidate order as KRIMP: first descending on support, second descending on itemset cardinality, and third lexicographically. Then, if compression is improved (9), the best code table so far is replaced with the new code table (10).

Keeping track of previous worlds in U - Not only the current world is taken into account, since this would mean that the code table would be adapted to each new world. By taking all previous worlds into account, the expected database can be estimated based on the sample mean. This is done by maintaining usages aggregated over all previously seen worlds in U, for each itemset in the code table. U is initiated in line 2 and maintained on line 13. This way, lossless compression of all worlds seen so far can be ensured, by defining

$$L(CT, U, D) = L(CT) + \frac{L(U \mid CT) + L(D \mid CT)}{k},$$

in which k is the number of samples considered so far, including D. Note that the encoded size of all previous worlds can be computed using only the aggregated usages in U; these usages together represent a complete cover of all previous worlds.

Pruning - If all worlds would be memorized and an itemset removed from the code table, a new cover of the data could be computed using the COVER algorithm. However, this is not the case, since previous worlds are not stored. Only aggregated usages for all $X \in CT$ are known through U. The solution is simple: remove itemset X from CT and cover X using the current CT, exactly as many times as its previous usage. This ensures a lossless encoding. For the current sample D, compute a new cover as usual. In each (pre or post) pruning step, a removal is accepted and made permanent iff $L(CT^{pruned}, U, D) \leq L(CT^*, U, D)$.

Post-accept pruning - After accepting an itemset, all itemsets are considered for removal one by one, using the methodology just described. (Similar to the pruning method used by KRIMP.)

Pre-accept pruning - This method is aimed at removing too specific itemsets from the code table. The intuition behind this is that itemsets that are specific for individual worlds may enter the code table, especially when k is still low. These itemsets may be in the way of more generic itemsets, that may be more appropriate for the "expected world". Therefore, when a candidate C is considered, all $X \in CT$ s.t. $X \supset C$ are considered for removal using the generic scheme outlined above.

8 Experiments

In this section we illustrate the theory presented so far with empirical evidence. According to the MDL principle, the best model is that model that compresses the data best. Therefore, the models that are induced by different methods can be judged by comparing total compressed sizes of both model and data, i.e. $L(D, CT)$. If one code table gives a smaller compressed size than another code table on the same dataset, we say that it is of higher quality. For ease of presentation, we will use compression ratio instead of compressed size, i.e. $L\% = \frac{L(D,CT)}{L(D,ST)} \times 100$.

In all experiments, a Laplace correction is applied to the obtained code tables, i.e. 1 is added to the usage of each itemset in the code table. This ensures that each code table can encode all possible transactions, as all singletons get a non-zero usage and thus a code.

8.1 Datasets. We next describe the datasets used in our experiments (basic properties are given in Table 1).

Synthetic - We generate two synthetic datasets, with $m = 16$ and $m = 20$, because for these m we can still materialize the $2^m \times m$ binary matrix $M(\mathcal{P})$. For each dataset, we first generate 10 random m-dimensional vectors as cluster centroids. Then, a probabilistic transaction is generated by picking one of these centroids uniformly at random and sampling a vector from a normal distribution ($\sigma = 1$) located at the centroid. This is repeated 1000 times to complete

Table 1: Dataset properties. Number of rows n, number of columns m, expected density (average of all probabilities in the matrix), and the used *minimum support* threshold are reported.

Dataset	*Properties*			
	n	m	E[density]	*minsup*
Biomine	1008200	1008200	0.00036%	0.01%
Synth-16m	1000	16	50%	0.1%
Synth-20m	1000	20	50%	0.1%
Paleo	871	453	8.3%	8%

the dataset.

Paleo - We also use the *Paleo* dataset [16], which contains information about fossils found on palaeontological sites in Europe. Although this is originally a binary dataset, it is well-known that only a very small percentage of all fossils is found. We argue that the probability that a species j once occurred at a certain site i depends on the distance to the closest site where j has been found. Therefore, the probability for (i, j) is computed as $\mathcal{P}_{i,j} = e^{-\sigma \times \texttt{mindist}}$, where σ is a parameter and `mindist` is the distance to the closest site where j occurs. If the closest site is i itself, the minimum distance is 0 and the probability 1. We use $\sigma = 5$, because this results in a dataset with an expected density that is neither too high nor too low.

Biomine - Finally, we use a recent snapshot of the *Biomine* dataset [26], which is a collection of biological interactions represented as a very large probabilistic graph with more than 1M nodes and 10M edges. Since interactions are undirected and labeled with probabilities, the graph can be represented as a square adjacency matrix where values are probabilities – a probabilistic database in which patterns represent typical node neighborhoods. Because interactions are undirected, some extra care is needed when sampling possible worlds: for all i, j, ensure that $D_{i,j} = D_{j,i}$.

8.2 Experiments on synthetic data. We first investigate the performance, in terms of compression, of the three presented methods on the synthetic datasets. With either $m = 16$ or $m = 20$, it is feasible to materialize the $2^m \times m$ binary matrix $M(\mathcal{P})$ of all possible transactions. Therefore we can apply *MED* and obtain a baseline code table that is a 'gold standard'. Next, we compress $M(\mathcal{P})$ with the code tables obtained with the other two methods, *SMM* and *SMUC*, so that we can compare the obtained compression ratios to that obtained with the gold standard.

As an additional test, we compute the average compression ratio on 20 newly sampled 'test' worlds. Since these samples were not used in the process of inducing the code table, this can be regarded as an independent test of how well any probable world is characterized.

For *SMM* and *SMUC*, 30 samples are used for code table induction. We empirically found this number of samples to be (more than) enough; using more samples did not result in better compression. More on this in the next subsection.

From the results shown in Table 2, it is clear that *SMM* approximates the gold standard *MED* very well. Of course, this is to be expected, as it follows from the theory stated in Section 6. Compression ratios obtained with *SMUC* are slightly worse, but the differences are small, especially when considering that *SMUC* only needs to store the latest sample.

All possible combinations of the 3 methods and the 2 pruning approaches are tested. Note that pre-pruning results are only available for *SMUC*; using it in the other algorithms would not make any difference, as a result of the candidate order. *MED* and *SMM* obtain the best results with post-accept pruning enabled, *SMUC* obtains the best results with both pre- and post-accept pruning enabled. Hence, both pruning methods allow the algorithms to explore good parts of the search space that are otherwise left untouched.

Comparing the best results on *Synthetic 16m*, we observe that ratios of 69.0, 69.1 and 70.8 are obtained with *MED*, *SMM* and *SMUC* respectively. *MED* is obviously the best as gold standard, while *SMM* is a very good approximation, as theory already showed. Finally, our novel incremental algorithm *SMUC* hardly stays behind, even though it builds the code table incrementally and only stores a single sample at a time.

The problem we stated in Section 4 is to find that code table that minimizes the expected compressed size of *any possible world*. Hence, it follows that the average compression ratio obtained on independently sampled worlds should be (almost) equal to the compression ratio obtained on the materialized database $M(\mathcal{P})$. Table 2 shows that is indeed the case; the ratios are almost identical. Therefore, we can use compression of newly sampled worlds as primary quality criterion for both the *Biomine* and the *Paleo* dataset, for which materializing the "expected database" $M(\mathcal{P})$ is clearly impossible.

8.3 Sampling real data. The next question is how many samples are needed for constructing a good code table, using either *SMM* or *SMUC*. For this purpose, let us consider Figure 1, which shows results obtained on *Paleo*. For these experiments, we vary the number of samples used for code table induction from 1 to 30.

Figure 1 shows the number of itemsets in the code table (bottom) and the average compression ratio $L\%$ on 20 sampled 'test' worlds (top). Note that in the

Table 2: **Synthetic** - Compression ratios obtained with code tables induced with *MED*, *SMM* and *SMUC*, with all possible combinations of enabling/disabling pre and post pruning.

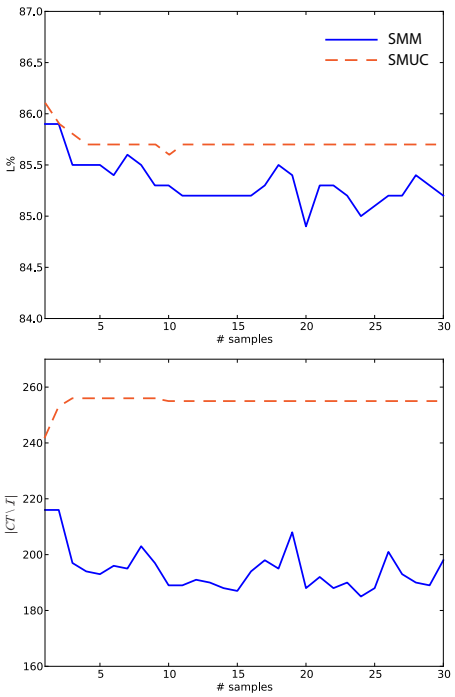
Synthetic 16m

Pruning		*L% on M(P)*			*avg L% on 20 Test Worlds*		
Pre	Post	*MED*	*SMM*	*SMUC*	*MED*	*SMM*	*SMUC*
		70.5	70.7	73.8	70.6	70.8	73.9
✓				72.5			72.5
	✓	69.0	69.1	71.4	69.0	69.1	71.3
✓	✓			70.8			70.8

Synthetic 20m

Pruning		*L% on M(P)*			*avg L% on 20 Test Worlds*		
Pre	Post	*MED*	*SMM*	*SMUC*	*MED*	*SMM*	*SMUC*
		69.9	69.7	73.3	69.9	69.7	73.4
✓				72.9			73.0
	✓	67.9	67.7	71.5	68.0	67.8	71.5
✓	✓			70.8			70.8

Figure 1: Results on *Paleo* with *SMM* and *SMUC*. Top: average *L*% on 20 test samples. Bottom: code table size.



lower plot, we use $|CT \setminus \mathcal{I}|$ instead of $|CT|$: we disregard the singletons, since these are always in the code table. Also note that both pruning methods are applied, since Table 2 showed that pruning helps to improve the final result.

For *SMM*, the plots show that although there is a tendency towards better compression for more samples, neither code table size nor compression ratio stabilize completely. This is due to the fact that there is quite some variation between the individual samples and the effect this has on the fairly small concatenated database. Since a completely new code table has to be built each time a sample is added to the concatenated database, small changes in the support of itemsets may have a large impact and thus robustness is low.

SMUC, on the other hand, converges nicely and quickly. The drawbacks are that it uses slightly more patterns and the achieved compression ratio is not as good. Still, it achieves a fairly good approximation. After a code table has converged (i.e. no patterns are added or removed), the best compression is attained on the test worlds. This makes a valid stopping criterion: it is easy to determine when enough samples have been seen. This is a large advantage with respect to *SMM*, where none of the available measures on 'training' samples, such as $|CT|$, $L\%$, and $L(CT)$, is predictive for compression performance on unseen 'test' worlds.

Table 3 shows results obtained on *Paleo*, with the two sampling-based methods and different pruning settings. To show that the *SMM* and *SMUC* code tables characterize the data distribution of any probable world as accurately as a code table induced from a specific world, we also compare *SMM* and *SMUC* against Krimp. To do so we sample 20 possible worlds, we mine them independently with Krimp, and we average the results. *SMM* and *SMUC* achieve almost the same compression ratios as Krimp, and often with less patterns. Here it is worth noting that the averaged Krimp results are based on 20 separate code tables, each of them tailored specifically to the same world it was also tested on, while the sampling methods need only one single code table that was learnt on training worlds, without knowing any of the test worlds.

Similar to the results on the synthetic data, *SMUC* approximates *SMM* very well in terms of compression and it has the advantage that it requires only very few samples to achieve this. Likewise, both pre- and post-pruning again contribute to a better compression, albeit that the differences are small. We will therefore apply both pruning methods in the remainder of the section.

Table 4 shows results obtained on the largest dataset, namely *Biomine*. Compression ratios larger than 100% may seem bad at first sight, but these are due to the large m, the sparsity of the data, and the Laplace correction that is applied to all code tables.

Table 3: **Paleo** - Code tables are induced with KRIMP, *SMM* and *SMUC*, with all possible combinations of enabling/disabling pre and post pruning. For KRIMP, each of 20 test samples is mined individually and average values over all samples are reported. For *SMM* and *SMUC*, average compression ratios obtained on the same 20 test sampled worlds are given (not used for code table induction). For *SMUC*, also the number of samples k required until convergence is displayed.

Method	Pre	Post	$\|CT \setminus \mathcal{I}\|$	$L\%$	k
KRIMP *(avg)*			400	86.7	
KRIMP *(avg)*		✓	234	84.7	
SMM			322	86.7	
SMM		✓	198	85.2	
SMUC			350	86.7	3
SMUC	✓		342	86.5	13
SMUC		✓	256	85.8	3
SMUC	✓	✓	255	85.7	10

Recall that compression is not the goal, but a means to select patterns. Therefore, the absolute ratios say something about the data, but for comparing the methods it is only important how the ratios relate to each other.

As before, 30 samples are used for *SMM*, while *SMUC* needs only 12 samples to converge. The results show that applying KRIMP to individual test worlds results in better compression than *SMM* and *SMUC*, which both build code tables without knowing these test worlds. This is due to the sparsity and variation between the individual worlds, making it harder to come up with a single small code table that performs well on any possible world. We can also observe that *SMUC* performs slightly better on this sparse data than *SMM*. More importantly, it requires an order of magnitude less memory: 252 Mb instead of a staggering 4849 Mb. Here, it clearly shows that considering samples one by one has important advantages to sampling, merging and mining many worlds at once.

8.4 Qualitative analysis of patterns. For a qualitative evaluation of the patterns mined from the probabilistic databases, we asked domain experts to further analyse the patterns obtained with *SMUC* (pre and post pruning enabled).

In contrast to regular frequent itemsets found in the original, rather sparse deterministic *Paleo* data, the patterns that our method finds in the probabilistic version consist of larger numbers of species. This is very useful for studying communities of animals, as patterns of 3-4 items are too small to represent interesting interactions between groups of species. Some of the high-usage patterns we find in *Paleo* consist of up to 8 distinct species, and indeed tend to represent ecologically meaningful communities that closely resemble those that experts have formed based on the existing fossil record. In particular, small and large mammals are mostly disjoint, and herbivores outnumber carnivores in all patterns. Furthermore, the patterns mostly contain herbivores, omnivores and a single predator, which according to domain experts is a realistic result. Also, some of the patterns are considered surprising by experts, such as one containing species from two different families of predators (a cat and a hyena that are rarely observed together), and another consisting of two different types of a horse genera and two hyena genera. Whether these represent artefacts of the data or true ecological phenomena is not clear though. [14]

Table 4: **Biomine** - Code table size, average compression ratio and memory usage. Code tables are induced with KRIMP, *SMM* and *SMUC*, with pre and post pruning enabled. For KRIMP, each of 20 test samples is mined individually and average values over all samples are reported. For *SMM* and *SMUC*, average compression ratios obtained on the same 20 test sampled worlds are given (not used for code table induction). Peak memory usage is also reported.

Algorithm	$\|CT \setminus \mathcal{I}\|$	$L\%$	*Memory (Mb)*
KRIMP (avg)	175	107.0	239
SMM	165	114.7	4849
SMUC	220	113.3	252

The entities in *Biomine* are genes, proteins and tissues, but also cellular components, molecular functions and biological processes. The latter three types are linked to the Gene Ontology project [1]. Three example patterns with high expected usage associate *DNA binding* with *transcription factory activity*, *zinc ion binding* with the *intracellular* component and *signal transduction* with *receptor activity*. For a biologist, these are typical, not surprising, but still meaningful patterns.

9 Conclusions

We introduce the problem of finding condensed and accurate pattern-based descriptions on uncertain data and analyze it from MDL perspective. We define the problem of mining the *expected optimal code table* and prove that it can be translated to the deterministic setting by materializing a $2^m \times m$ binary matrix. When this is infeasible, we rely on sampling. We show that the problem can be accurately approximated by sampling possible worlds and concatenating these into a single database. This approach has two drawbacks though: it requires the number of samples as input and imposes

large memory requirements. Therefore we introduce a third method that incrementally samples a world, mines it and improves the current global description, until the description itself converges to a stable state. This way the memory requirements are kept as low as when mining a deterministic dataset, thus allowing to scale to very large probabilistic datasets.

Experiments in paleontology and bioinformatics domains show that from very large probabilistic matrices, we can extract small sets of interesting and meaningful patterns that accurately characterize the data distribution of any probable world.

Acknowledgements The authors would like to thank Hannu Toivonen, Jussi Eronen, and Arno Siebes for their kind help with the data and/or giving helpful comments. Matthijs van Leeuwen is financially supported by the Netherlands Organisation for Scientific Research (NWO), under project number 612.065.822.

References

[1] Gene ontology: tool for the unification of biology. The Gene Ontology Consortium. *Nature Genetics*, 25(1):25–29, 2000.

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

[3] C. C. Aggarwal, Y. Li, J. Wang, and J. Wang. Frequent pattern mining with uncertain data. In *KDD*, pages 29–38, 2009.

[4] C. C. Aggarwal and P. S. Yu. A framework for clustering uncertain data streams. In *ICDE*, pages 150–159, 2008.

[5] T. Bernecker, H.-P. Kriegel, M. Renz, F. Verhein, and A. Züfle. Probabilistic frequent itemset mining in uncertain databases. In *KDD*, pages 119–128, 2009.

[6] B. Bringmann and A. Zimmermann. One in a million: picking the right patterns. *Knowl. Inf. Syst.*, 18(1):61–81, 2009.

[7] T. Calders, C. Garboni, and B. Goethals. Efficient pattern mining of uncertain data with sampling. In *PAKDD (1)*, pages 480–487, 2010.

[8] V. Chandola and V. Kumar. Summarization – compressing data into an informative representation. *Knowl. Inf. Syst.*, 12(3):355–378, 2007.

[9] C. K. Chui and B. Kao. A decremental approach for mining frequent itemsets from uncertain data. In *PAKDD*, pages 64–75, 2008.

[10] C. K. Chui, B. Kao, and E. Hung. Mining frequent itemsets from uncertain data. In *PAKDD*, pages 47–58, 2007.

[11] G. Cormode and A. McGregor. Approximation algorithms for clustering uncertain data. In *PODS*, pages 191–200, 2008.

[12] N. N. Dalvi, C. Ré, and D. Suciu. Probabilistic databases: diamonds in the dirt. *Commun. ACM*, 52(7):86–94, 2009.

[13] N. N. Dalvi and D. Suciu. Efficient query evaluation on probabilistic databases. In *VLDB*, pages 864–875, 2004.

[14] J. Eronen. Personal communication, 2010.

[15] C. Faloutsos and V. Megalooikonomou. On data mining, compression and kolmogorov complexity. *Data Min Knowl Discov*, 15(1):3–20, 2007.

[16] M. Fortelius. Neogene of the old world database of fossil mammals (now). June 2009. `http://www.helsinki.fi/science/now/`, 2009.

[17] P. D. Grünwald. Minimum description length tutorial. In P. Grünwald and I. Myung, editors, *Advances in Minimum Description Length*. MIT Press, 2005.

[18] H. Heikinheimo, J. Vreeken, A. Siebes, and H. Mannila. Low-entropy set selection. In *Proceedings of the SDM'09*, pages 569–579, 2009.

[19] W. Hoeffding. Probability inequalities for sums of bounded random variables. *Journal of the American Statistical Association*, 58(301):13–30, March 1963.

[20] B. Kao, S. D. Lee, D. W. Cheung, W.-S. Ho, and K. F. Chan. Clustering uncertain data using voronoi diagrams. In *ICDM*, pages 333–342, 2008.

[21] A. J. Knobbe and E. K. Y. Ho. Pattern teams. In *ECML PKDD*, pages 577–584, 2006.

[22] M. van Leeuwen and A. Siebes. Streamkrimp: Detecting change in data streams. In *ECML PKDD*, pages 672–687, 2008.

[23] M. van Leeuwen, J. Vreeken, and A. Siebes. Compression picks item sets that matter. In *ECML PKDD 2006*, pages 585–592, 2006.

[24] M. Li and P. Vitányi. *An introduction to Kolmogorov complexity and its applications*. Springer-Verlag, 1993.

[25] J. Ren, S. D. Lee, X. Chen, B. Kao, R. Cheng, and D. W.-L. Cheung. Naive bayes classification of uncertain data. In *ICDM*, pages 944–949, 2009.

[26] P. Sevon, L. Eronen, P. Hintsanen, K. Kulovesi, and H. Toivonen. Link discovery in graphs derived from biological databases. In *DILS*, 2006.

[27] A. Siebes, J. Vreeken, and M. van Leeuwen. Item sets that compress. In *SDM*, pages 393–404, 2006.

[28] L. Sun, R. Cheng, D. W. Cheung, and J. Cheng. Mining uncertain data with probabilistic guarantees. In *KDD*, pages 273–282, 2010.

[29] N. Tatti and J. Vreeken. Finding good itemsets by packing data. In *ICDM*, pages 588–597, 2008.

[30] S. Tsang, B. Kao, K. Y. Yip, W.-S. Ho, and S. D. Lee. Decision trees for uncertain data. In *ICDE*, pages 441–444, 2009.

[31] J. Vreeken, M. van Leeuwen, and A. Siebes. Characterising the difference. In *SIGKDD*, pages 765–774, 2007.

[32] J. Wang and G. Karypis. On efficiently summarizing categorical databases. *Knowl. Inf. Syst.*, 9(1):19–37, 2006.

[33] E. Zimányi and A. Pirotte. Imperfect information in relational databases. In *Uncertainty Management in Information Systems*, pages 35–88. 1996.