Data Min Knowl Disc (2013) 26:130–173
DOI 10.1007/s10618-011-0246-6

# Summarizing categorical data by clustering attributes

**Michael Mampaey · Jilles Vreeken**

Received: 27 April 2011 / Accepted: 16 November 2011 / Published online: 26 November 2011


**Abstract** For a book, its title and abstract provide a good first impression of what to expect from it. For a database, obtaining a good first impression is typically not so straightforward. While low-order statistics only provide very limited insight, downright mining the data rapidly provides too much detail for such a quick glance. In this paper we propose a middle ground, and introduce a parameter-free method for constructing high-quality descriptive summaries of binary and categorical data. Our approach builds a summary by clustering attributes that strongly correlate, and uses the Minimum Description Length principle to identify the best clustering—without requiring a distance measure between attributes. Besides providing a practical overview of which attributes interact most strongly, these summaries can also be used as surrogates for the data, and can easily be queried. Extensive experimentation shows that our method discovers high-quality results: correlated attributes are correctly grouped, which is verified both objectively and subjectively. Our models can also be employed as surrogates for the data; as an example of this we show that we can quickly and accurately query the estimated supports of frequent generalized itemsets.

**Keywords** Attribute clustering · MDL · Summarization · Categorical data

Responsible editor: M.J. Zaki.

The research described in this paper builds upon and extends the work appearing in ECML PKDD'10 as Mampaey and Vreeken (2010).

M. Mampaey (✉) · J. Vreeken
Advanced Database Research and Modelling, Department of Mathematics and Computer Science, University of Antwerp, Antwerp, Belgium
e-mail: michael.mampaey@ua.ac.be

J. Vreeken
e-mail: jilles.vreeken@ua.ac.be

## 1 Introduction

When handling a book, and wondering about its contents, we can simply start reading it from A to Z. In practice, however, to get a good first impression we usually first consult the summary. For a book, this can be anything from the title, abstract, or blurb, up to simply paging through it. The common denominator here is that a summary quickly provides high-quality and high-level information about the book. Occasionally a summary may already offer us exactly what we were looking for, but in general we just expect to get sufficient insight to determine what the book contains, and whether we need to read it further.

When handling a database, on the other hand, and wondering about its contents and whether (or how) we should analyze it, it is quite hard to get a good first impression. Naturally, we can inspect the schema of the database or look at the labels of the attributes. However, this does not tell use what is *in* the database. Basic statistics only help to a limited extent. For instance, first order statistics can tell us which values of an attribute occur, and how often. While sometimes this may be enough information, typically we would like to see in a bit more detail what the data looks like. However, for binary and categorical databases, further basic statistics are not readily available. Ironically, for these types of data this means that even if the goal is only to get a first impression, in order to acquire this we will have to analyze the data in much more detail. For non-trivially sized databases especially, this means investing far more time and effort than we should at this stage of the analysis.

Having a good first impression of the data is highly important when analyzing data, as data mining is an essentially iterative process (Hanhijärvi et al. 2009), where each step in the analysis is aimed at obtaining new insight. Insight that, in turn, determines what other results we would find interesting. And, hence, determines how to proceed in order to extract further knowledge from the data. As such, a good summary allows us to make a well-informed decision on what basic assumptions to make and how to start mining the data.

To this end, we here propose a simple and parameter-free method for providing high-quality summary overviews for categoric data, including binary transaction data. These summaries provide insight in which attributes are most correlated, as well as in what value configurations they occur. They are probabilistic models of the data that can be queried fast and accurately, allowing them to be used instead of the data. Further, by showing which attributes interact most strongly, these summaries can aid in selecting or constructing features. In short, like a proper summary, they provide both a good first impression and can be used as a surrogate.

We obtain these summaries by clustering attributes that interact strongly, such that attributes from different clusters are more or less independent. We employ the Minimum Description Length (MDL) principle to identify the best clustering; so eliminating the need for the user having to specify any parameters: the best summary is the attribute clustering that describes the data best. Furthermore, our clustering approach does not require a distance measure between attributes.

As an example of a summary, and how it provides basic insight, consider Fig. 1, in which we depict the summary of a large hypothetical categorical database. A summary provides information at two levels of detail. First, it tells us which groups of

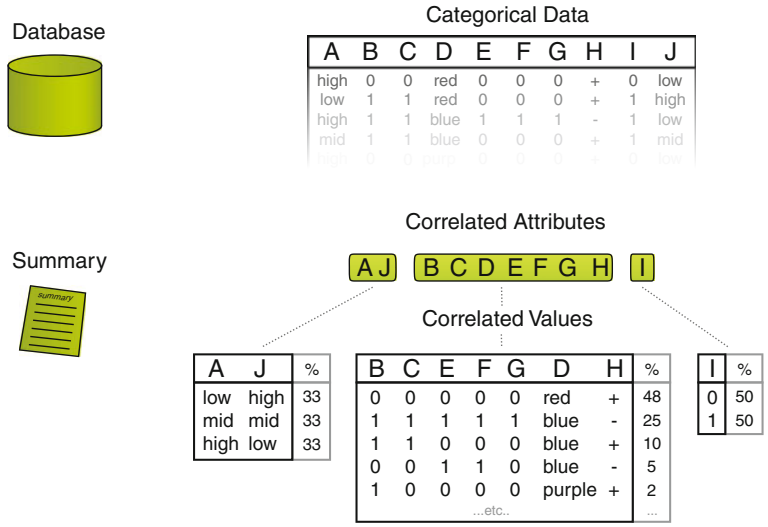


**Fig. 1** Example of a database summary. A summary shows which groups of attributes most strongly interact, as well as which value combinations these occur in the database. For presentational clarity, we only show the top-5 most frequent value combinations for $B$–$H$ here

attributes correlate most strongly. Here, we see that attributes $A$ and $J$ are grouped together, as are attributes $B$–$H$, and that $I$ forms a singleton group. For these groups, by the MDL principle, we know that the attributes within them interact strongly, while between the groups the attributes can be considered virtually independent—because if an attribute provides information on another, we would have saved bits by considering them together. As such, here, we learn that $I$ does not interact much with any of the other attributes.

At the second level, summaries allow quick inspection of the attribute-value distributions within the groups. In the example, we see that $A$ and $J$ are each other's inverse, and that attributes $B$–$H$ have two strongly prevalent value configurations, after which the frequencies drop off rapidly. While a summary contains the frequencies of all attribute-value combinations per group, typically one is most interested in the most prevalent, and hence we here only show the top-5.

We will investigate and analyze summaries obtained from real data in Sect. 7. Extensive experimentation shows our method provides high-quality results: correlated attributes are correctly grouped, representative features are identified, and the supports of frequent generalized itemsets are closely approximated in very short time. Randomization shows that our approach models relevant structure in the data, providing information far beyond simple first order statistics.

To the best of our knowledge, there currently do not exist light-weight data analysis methods that can easily be used for summary purposes. Instead, for binary and categorical data, a standard approach is to first mine for frequent itemsets, the result of which quickly grows to many times the size of the original database. Consequently, many proposals exist that focus on reducing or summarizing these sets of frequent patterns. That is, they choose groups of representative itemsets such that the information

in the complete pattern set is maintained as well as possible. In this paper, we do not summarize the outcome of an analysis (i.e., a set of patterns), but instead provide a summary that can be used to decide how to further analyze the data.

Existing proposals for data summarization, such as KRIMP (Siebes et al. 2006) and Summarization (Chandola and Kumar 2005), provide highly detailed results. Although this has obvious merit, analyzing these summaries consequently also requires significant effort. Our method shares the approach of using compression to find a good summary. However, we do not aim to find a group of descriptive itemsets. Instead, we look at the data on the level of attributes, and aim to optimally group those attributes that interact most strongly. In this regard, our approach is related to mining low-entropy sets (Heikinheimo et al. 2007), sets of attributes which identify strong interactions in the data. An existing proposal to summarize data by low-entropy sets, LESS (Heikinheimo et al. 2009), requires a collection of low-entropy sets as input, and the resulting models are not probabilistic in nature, nor can they easily be queried. For a more complete discussion of related work, please refer to Sect. 6.

A primary version of our approach was published as Mampaey and Vreeken (2010). Here, we generalize our approach from binary to categoric data, we thoroughly discuss the theory and choices, and we give a refined variant of our encoding that takes insights from modern MDL theory into account. Further, we provide a more extensive experimental evaluation of our method, using 19 benchmark and real-world datasets. In particular, we give a detailed evaluation of the heuristic choices made, comparing between the different search strategies and encodings, and show that the discovered summaries model relevant structure in the data. Furthermore, we investigate the speed at which our summaries can be queried.

The road map of this paper is as follows. First, Sect. 2 introduces notation and preliminaries. Section 3 formalizes the problem, and discusses how to identify good summaries. In Sect. 4 we present our method for discovering good attribute clusterings. Section 5 discusses potential alternative search strategies. Related work is discussed in Sect. 6. We experimentally evaluate our method in Sect. 7. Lastly, we round up with discussion in Sect. 8 and conclude the paper in Sect. 9.

## 2 Preliminaries

We denote the set of all attributes by $\mathcal{A} = \{a_1, \ldots, a_n\}$. The set of possible values $\{v_1, v_2, \ldots\}$ that an attribute $a$ can obtain is called the domain of $a$, written as $dom(a)$. In this paper we assume that all attributes are categorical, i.e., that they have a discrete and finite domain: $|dom(a)| \in \mathbb{N}$. Note that binary attributes are simply categorical attributes whose domain is equal to $\{0, 1\}$. As such, any binary dataset can be regarded as a categorical dataset. Hence, unless specified otherwise, whenever we speak about categorical datasets, the same holds for binary datasets.

The domain of a set of attributes $X$ is the Cartesian product of the domains of its individual attributes: $dom(X) = \prod_{a \in X} dom(a)$. An item is a attribute-value combination, that is, a pair $(a = v)$, where $a \in \mathcal{A}$ and $v \in dom(a)$. Analogously, an itemset is a pair $(X = v)$, where $X \subseteq \mathcal{A}$ is an attribute set, and $v \in dom(X)$ is a vector of

length $|X|$. When $\mathcal{A}$ consists of binary attributes and $v = 1$ is a vector of ones, the itemset $(X = 1)$ is often written simply as $X$.

A categorical dataset $D$ is a bag of transactions (or tuples) $t$, which are vectors of length $n$ containing one value for each attribute. A transaction $t$ is said to contain an itemset $(X = v)$, denoted as $(X = v) \subseteq t$, if for all attributes $a \in X$ it holds that $t_a = v_a$. The support of $(X = v)$ is the number of transactions in $D$ that contain it:

$$supp(X = v) = |\{t \in D \mid (X = v) \subseteq t\}|.$$

The frequency of $(X = v)$ is defined as its support relative to the number of transactions in $D$ : $fr(X = v) = supp(X = v)/|D|$.

The entropy of an attribute set $X$ is defined as

$$H(X) = - \sum_{v \in dom(X)} fr(X = v) \log fr(X = v),$$

where the base of the logarithm is 2, and by convention $0 \log 0 = 0$.

## 3 Summarizing data by clustering attributes

In this section we formally introduce our approach. We start with a brief primer on the MDL principle, and after defining what an attribute clustering is, we show how we can use MDL to identify the best clustering. We then formalize the problem, and discuss the search space. Finally, we discuss a more refined encoding that takes cues from modern MDL theory.

### 3.1 MDL, a brief primer

The MDL principle (Rissanen 1978), like its close cousin Minimum Message Length (MML) (Wallace 2005), is a practical version of Kolmogorov Complexity (Li and Vitányi 1993). All three embrace the slogan *Induction by Compression*. The MDL principle can be roughly described as follows.

Given a set of models $\mathcal{M}$, the best model $M \in \mathcal{M}$ is the one that minimizes

$$L(M) + L(D \mid M)$$

in which

- $L(M)$ is the length, in bits, of the description of $M$, and
- $L(D \mid M)$ is the length, in bits, of the description of the data encoded with $M$.

This is called two-part MDL, or *crude* MDL. This stands opposed to *refined* MDL, where model and data are encoded together (Grünwald 2007). We use two-part MDL because we are specifically interested in the model: the attribute clustering that yields the best description length. Further, although refined MDL has stronger theoretical foundations, it cannot be computed except for some special cases.

To use MDL, we have to define what our set of models $\mathcal{M}$ is, how a model $M$ describes a database, and how all of this is encoded in bits. Below, after giving some further intuition on MDL, we will first discuss a two-part MDL encoding for our summaries, as well as a variant that draws from insights from refined MDL.

The reason we employ the MDL principle, is that it gives us a well-founded approach for selecting the model that best balances the complexity of the model and the data. That is, models that encode the data concisely, i.e., for which $L(D \mid M)$ is low, are considered to be good models, because they are able to exploit, and ultimately describe, the structure of the data. In practice, this term often corresponds to the negative log-likelihood of the model.

However, it is possible to come up with an arbitrarily complex model that describes the data arbitrarily well—in the most extreme case the 'model' just states the data verbatim, which is clearly a form of overfitting. Therefore, a model that has a long description, i.e., for which $L(M)$ is large, is likely to overfit the data, and hence considered to be not so good. Models that can be described succinctly, on the other hand, are less prone to overfitting, and additionally also tend to be easier to interpret. Of course, models that are overly simple will not do a good job describing the data.

The MDL principle provides an appropriate balance between these extremes, by taking both model fit and model complexity into account. MDL is tightly linked to lossless data compression—the best model is the model that compresses the data best. It is also often formulated as a sender/receiver framework, where a sender wants to communicate the data to a receiver, in an as compact as possible message. The sender encodes both the model and the data, and transmits them to the receiver, who decodes them to obtain the original data. The goal of this paper is purely to model and summarize the data, not to actually compress or transmit it. However, it is useful to keep this analogy in mind.

Given this basic intuition, let us next formalize how to lossless encode a model and how to encode the data given that model.

### 3.2 MDL for attribute clustering

The main idea for our data summaries are *attribute clusterings*. Therefore, we first formally define the concept of an attribute clustering.

**Definition 1** An attribute clustering $\mathcal{C} = \{A_1, \ldots, A_k\}$ of a set of attributes $\mathcal{A}$ is a partition of $\mathcal{A}$, that is,

1. every attribute belongs to a cluster: $\bigcup_{i=1}^{k} A_i = \mathcal{A}$,
2. all clusters are pairwise disjoint: $\forall i \neq j : A_i \cap A_j = \emptyset$,
3. there are no empty clusters: $\forall A_i \in \mathcal{C} : A_i \neq \emptyset$.

Informally speaking, an attribute clustering is simply a partition of the set of attributes $\mathcal{A}$. In order to give the user a bit more information, for each cluster we also include the distribution of the attribute-value occurrences in the data.

Given this definition, we can formalize how we actually encode our models and data, such that we can apply the MDL principle to distinguish good summaries from bad ones. We start by specifying how we encode our models, since then the encoding

of the data follows straightforwardly. In our case, a model is a partition of the attributes, $\mathcal{A}$, together with descriptions of which value combinations occur how often, per partition, i.e., the distributions within the clusters.

### 3.2.1 Encoding the partition

Let us first describe how we can encode an attribute partition. Recall that the number of unique partitions for a set of $n$ attributes is given by the well-known Bell number, denoted $B_n$, which can be computed using a simple recursive formula:

$$B_{n+1} = \sum_{k=0}^{n} \binom{n}{k} B_k,$$

with base case $B_0 = 1$. Now, using a basic approach in information theory, we can fix a canonical order on the set of partitions, which allows us to enumerate all possible partitions. More importantly, this allows us to uniquely and efficiently identify a particular partition using $\log B_n$ bits. This is called a model-to-data code (Vereshchagin and Vitanyi 2004), and is the encoding we use to identify the partition of $\mathcal{A}$ of a given attribute clustering.

### 3.2.2 Encoding the code tables

Next, we formalize how to encode the joint distribution of clusters by using *code tables*. A code table is a simple two-column table that has an entry for each possible value $v$ from the domain of a cluster $A_i$. The left-hand column contains the value, and the right-hand column the corresponding code. By making sure the used codes are uniquely decodable, we can straightforwardly and lossless translate between a code and the associated values. We give an example code table in Table 1. Note that the frequencies of the value occurrences are not explicitly included in the code table, we will show below that we can derive these exactly from the lengths of the codes in the right-hand column. Also, if a value combination does not occur in the data (when its frequency is zero), it is omitted from the code table, and it does not get a code.

**Table 1** Example of a code table $CT_i$ for a cluster $A_i = \{a, b, c\}$ containing three binary attributes

| Code table | | | | $L(code(v))$ | $fr(A = v)$ (%) |
|---|---|---|---|---|---|
| a | b | c | $code(A = v)$ | | |
| 1 | 1 | 1 | 0 | 1 bits | 50.00 |
| 1 | 1 | 0 | 10 | 2 bits | 25.00 |
| 1 | 0 | 1 | 110 | 3 bits | 12.50 |
| 0 | 1 | 0 | 1110 | 4 bits | 6.25 |
| 0 | 0 | 0 | 1111 | 4 bits | 6.25 |

The (optimal) codes depicted here are included for illustration purposes, in practice we are only interested in their lengths, $L(code(v))$. Assuming the database contains 256 transactions, the description length of this code table is 44 bits

We encode the entries of the left-hand column as follows. Again using a model-to-data code, we can identify one of the values of an attribute $a$ in $\log |dom(a)|$ bits, given that the size of the domain of $a$ is $|dom(a)|$. Therefore, the description length of a value $v$ of a set of attributes $A_i$ is equal to

$$\sum_{a \in A_i} \log |dom(a)| = \log |dom(A_i)|.$$

The entries in the right-hand column of a code table are the codes that will be used to encode the data. Clearly, these codes should be as efficient as possible, which is why we employ an optimal prefix code. A prefix code (or, confusingly also known as a prefix-free code) is a code that can be uniquely decoded without requiring a prefix. A well-known result from information theory (Shannon 1948; Cover and Thomas 2006) states that a prefix code is *optimal* when for all $v \in dom(A_i)$

$$L(code(A_i = v)) = -\log fr(A_i = v).$$

The choice of one particular optimal coding scheme over another (for instance, Huffman coding (Cover and Thomas 2006), which we use to obtain the codes in Table 1), is irrelevant: recall that in MDL we are not concerned about actual materialized codes, but only want to measure complexity. Hence, we are only interested in the *lengths* of the theoretical optimal code the equation above gives us.

So, the encoded length of the right-hand side column of a code table is basically the sum of each of the code lengths. However, in order to be able to uniquely decode a code table, we need to take one further element into account: we need to know when a bit representation of a code ends. Therefore, before a code is given, we specify its length in a fixed number of bits; since the frequency of an itemset that occurs in the data is at least $1/|D|$, its corresponding code length is at most $\log |D|$, and hence the length of the code can be stated in $\log \log |D|$ bits.

Now, a binary string so-representing a distribution, viz. code table, can unambiguously be decoded: by the association between code lengths and frequencies we can derive the frequencies of the values on the left-hand side by regarding the code lengths found on the right-hand side. By adding these frequencies together, we know the distribution is fully specified when this sum equals one. Therefore, the description length of the code table $CT_i$ of an attribute cluster $A_i$ can be computed as

$$L(CT_i) = \sum_{\substack{v \in dom(A_i) \\ fr(A_i = v) \neq 0}} \log |dom(A_i)| + \log \log |D| - \log fr(A_i = v).$$

Using this encoding, clusters that have many value instantiations that are distributed irregularly, will require many bits. Similarly, attribute clusters with few, evenly distributed values, are much cheaper. In other words, we favor simple distributions.

### 3.2.3 Encoding the data

Now that we know how to encode a clustering, we can discuss how to determine $L(D \mid \mathcal{C})$, the length of the encoded description of $D$ given a clustering $\mathcal{C}$. This is done straightforwardly. First, each tuple $t \in D$ is partitioned according to $\mathcal{C}$. We then encode a tuple by replacing the value in each part by the corresponding code in the code tables. Since an itemset $(A = v)$ occurs in the data $|D|fr(A = v)$ times, the encoded size of $D$ restricted to a single cluster $A_i$ equals

$$\begin{aligned} L(D_{A_i} \mid \mathcal{C}) &= -\sum_{t \in D} \log fr(A = t_A) \\ &= -|D| \sum_{v \in dom(A)} fr(A = v) \log fr(A = v) \\ &= |D| H(A_i). \end{aligned}$$

That is, the description length of the data with respect to an attribute cluster is proportional to its entropy, which is a measure of complexity.

Combining all of the above, the definition of the total encoded size is as follows.

**Definition 2** The description length of a categorical dataset $D$ using an attribute clustering $\mathcal{C} = \{A_1, \ldots, A_k\}$ of size $k$ is defined as

$$L(\mathcal{C}, D) = L(\mathcal{C}) + L(D \mid \mathcal{C}),$$

where

$$\begin{cases} L(D \mid \mathcal{C}) = |D| \sum_{i=1}^{k} H(A_i) \\ L(\mathcal{C}) \quad = \log B_n + \sum_{i=1}^{k} L(CT_i) \\ L(CT_i) \quad = \sum\limits_{\substack{v \in dom(A_i) \\ fr(A_i = v) \neq 0}} \log |dom(A_i)| + \log\log |D| - \log fr(A_i = v) \end{cases}$$

Note that in the middle line, we can simply take the sum over all code table lengths, i.e., we do not need to indicate how many code tables there are (this is captured in the $\log B_n$ term), nor do we need to separate them with additional bits, since the descriptions of the code tables are self-terminating, as explained above.

Further, remark that we implicitly make a basic assumption. To use the above definition, we assume that the number of attributes, their domains (their sizes in particular), and the number of transactions in the database are known beforehand to both sender and receiver. The reason not to include these in our formalization is simple: we are aiming to summarize a single given dataset. Clearly, these properties are constant over all models we would consider for one dataset, and hence, including these would increase the total description length only by a constant term, independent of the actual data instance and model, which makes no difference when comparing different clusterings. If for one reason or another it would be required to explicitly include the cost

of these values into the total description length, one could do so by using a Universal Code for integers (Rissanen 2007).

### 3.3 Problem statement

Our goal is to discover a summary of a binary or categorical dataset, in the form of a partitioning of the attributes of the data; separate attribute groups should be relatively independent, while attributes within a cluster should exhibit strong interaction.

Formally, the problem we address is the following. Given a database $D$ over a set of categorical attributes $\mathcal{A}$, find the attribute clustering $\mathcal{C}^*$ that minimizes $L(\mathcal{C}, D)$,

$$\mathcal{C}^* = \arg\min_{\mathcal{C}} L(\mathcal{C}, D) = \arg\min_{\mathcal{C}} L(\mathcal{C}) + L(D \mid \mathcal{C}).$$

By this problem statement, we identify the optimal clustering by MDL. Note that we do not require the user to specify any parameters, e.g., a predetermined number of clusters $k$. Essentially, this is done automatically, as $k$ follows from the clustering that has the shortest description.

### 3.4 Search space

The search space to be considered for our problem is rather large. The total number of possible partitions of a set of $n$ attributes equals the Bell number $B_n$, which is at least $\Omega(2^n)$. Therefore we cannot simply enumerate and test all possible partitions, except for the most trivial cases. We must exploit the structure of the search space somehow, in order to efficiently arrive at a good clustering.

The *refinement* relation of partitions naturally structures the search space into a lattice. A partition $\mathcal{C}$ is said to refine a partition $\mathcal{C}'$ if for all $A \in \mathcal{C}$ there exists an $A' \in \mathcal{C}'$ such that $A \subseteq A'$. The transitive reduction of the refinement relation corresponds to the merger of two clusters into one (or conversely, splitting a cluster into two nonempty parts). As such, the search space lattice can be traversed in a stepwise manner.

The maximal, most refined clustering contains a singleton cluster for each individual attribute. We call this the *independence clustering*, denoted $\mathcal{I}$, since it corresponds to the independence distribution. On the other hand, the least refined, most coarse partition consists of only one cluster containing all attributes, which corresponds to the full joint distribution.

Note that $L(\mathcal{C}, D)$ is not (strictly or weakly) monotonically increasing or decreasing with respect to refinement, which would make determining the optimum trivial—this would require there would exist at least one monotonic path between $\mathcal{I}$ and $\{\mathcal{A}\}$. As far as we know, other useful properties that could be exploited to identify the optimal clustering efficiently, such as convertibility, also do not apply to this setting. Hence, we will have to resort to heuristics. In this case, we will employ a greedy hierarchical clustering strategy, the details of which are discussed in Sect. 4.

### 3.5 Measuring similarities between clusters

Based on the definition of $L(\mathcal{C}, D)$ above, we can derive a similarity measure between clusters that will turn out to be useful later on. Let $\mathcal{C}$ be an attribute clustering and let $\mathcal{C}'$ be the result of merging two clusters $A_i$ and $A_j$ in $\mathcal{C}$, the merged cluster being denoted $A_{ij} = A_i \cup A_j$. Hence, $\mathcal{C}$ is a refinement of $\mathcal{C}'$. Then the difference between the description lengths of $\mathcal{C}$ and $\mathcal{C}'$ defines a similarity measure between $A_i$ and $A_j$.

**Definition 3** We define the similarity of two clusters $A_i$ and $A_j$ in a clustering $\mathcal{C}$ as

$$CS_D(A_i, A_j) = L(\mathcal{C}, D) - L(\mathcal{C}', D),$$

where $\mathcal{C}' = \mathcal{C} \setminus \{A_i, A_j\} \bigcup \{A_i \cup A_j\}$. Whenever $D$ is clear from the context, we simply write $CS(A_i, A_j)$.

If $A_i$ and $A_j$ are highly correlated, then their merger results in a better clustering with a lower description length, and hence their similarity is positive. Otherwise, if the clusters are more or less independent, their merger increases the description length, and their similarity is negative. The fact that *CS* expresses a similarity is further illustrated by the following proposition, which allows us to calculate cluster similarity without having to compute the total description length of a clustering.

**Proposition 1** *Let $\mathcal{C}$ be an attribute clustering of $\mathcal{A}$, with $A_i, A_j \in \mathcal{C}$, and let $D$ be a categorical dataset. Then*

$$CS_D(A_i, A_j) = |D| I(A_i, A_j) + \Delta L(CT),$$

*where*

$$I(A_i, A_j) = H(A_i) + H(A_j) - H(A_{ij})$$

*is the mutual information between $A_i$ and $A_j$, and*

$$\Delta L(CT) = L(CT_i) + L(CT_j) - L(CT_{ij}).$$

Proposition 1 shows that we can decompose cluster similarity into a mutual information term, and a term expressing the difference in code table description length. Both of these values are high when $A_i$ and $A_j$ are highly correlated, and low when they are more or less independent. It is interesting to note that *CS* is a local measure, that is, it only depends on $A_i$ and $A_j$, and is not influenced by the other clusters in $\mathcal{C}$.

### 3.6 Canonical description length

It can be useful to compare the description length $L(\mathcal{C}, D)$ to a baseline description of the data; a description that does not use a model, but simply communicates the data 'as is'. However, there are many different ways to encode a database, all with different

description lengths. A very natural approach is to use an encoding assuming a uniform distribution of the values. In this encoding, every value of an attribute $a$ receives a code with the smallest possible number of bits under this assumption, namely $\log |dom(a)|$.

**Definition 4** The *canonical description length* of a dataset $D$ is defined as

$$L_c(D) = |D| \sum_{a \in \mathcal{A}} \log |dom(a)| = |D| \log |dom(\mathcal{A})|.$$

For example, if $D$ is a binary dataset, its canonical description length is equal to $|D||\mathcal{A}|$ bits, i.e., each entry requires exactly one bit.

It can be argued that if for some model $\mathcal{C}$, we observe that $L(\mathcal{C}, D) \ll L_c(D)$, we can safely say that $\mathcal{C}$ better captures or explains the structure in the data, and hence $\mathcal{C}$ can be considered a good model. On the other hand, if $D$ mostly just consists of random noise (that is, $D$ does not exhibit any structure), then for any model $\mathcal{C}$ we will find that $L(\mathcal{C}, D) \geq L_c(D)$; if such is the case, we may conclude the data does not exhibit any structure that our model can fit.

### 3.7 Refining the encoding

The description length of an attribute clustering for a certain dataset as described above, is often called *two-part* or *crude* MDL, since it separately encodes the model and the data. Refined MDL (Grünwald 2007) is an improvement of two-part MDL that does not explicitly encode model and data separately. Rather, it uses a so-called Universal Code to describe the data. A universal code is a code that we can employ without having any prior information, and for which the lengths of the code words are within a constant factor of the optimal codes. As such, using universal codes avoids the potential bias that can occur in two-part MDL.

Although refined MDL has stronger theoretical foundations, it cannot be computed except for some special cases; those for which a universal code is known. As only a very limited number of universal codes is known, for distributions of relatively simple objects, the practical application of refined MDL is limited. However, we can refine our encoding somewhat by using a notion from modern MDL theory (Grünwald 2007); we can employ *prequential* coding.

By prequential coding, we are not explicitly transmitting the codes that will be used to encode the data—and so, we lose any potential bias that the explicit encoding of those codes might give. Rather than using explicit fixed code words, we encode the data using an implicitly assumed distribution on the data, and iteratively update this distribution (and hence, the codes) after every transmitted/received code word. As such, by each update the distribution changes slightly, and hence so does the underlying code table, and thus the encoding of the data that is to be transmitted.

It may seem that by not explicitly including a cost for using long code words, we do not penalize against this. However, the penalty is actually hidden in the description of the data, since there is always an overhead incurred by using a code table that does not exactly represent the distribution of the data, i.e., a code table that is suboptimal.

Before we discuss the size of this penalty, let us first specify how we can employ prequential coding in our setting. When we start transmitting the data, instead of using the actual attribute-value frequencies of the data, we start without such prior knowledge and use a uniform distribution over these frequencies, by assigning some constant occurrence count $c$ to each possible value in the domain of an attribute set. Then, we transmit the first value by encoding it using an optimal code derived from this usage distribution, as defined in Sect. 3.2. After this value is transmitted, we adjust the distribution by incrementing the occurrence count of that particular value by 1. Clearly, this changes the usage distribution, and hence the codes in the code table must be recomputed in order to remain optimal. We simply repeat this process iteratively for every tuple, until we have transmitted all of the data—after which, if we disregard $c$, the codes in the code table are the same as when we would have transmitted them explicitly.

The choice of the initial count $c$ has an influence on the overhead. If $c$ is taken very large, updating the code table with new data has little effect. For $c \to \infty$, the overhead tends to $KL(\mathbf{u} \,\|\, fr)$, i.e., the Kullback–Leibler divergence between $fr$ and the uniform distribution $\mathbf{u}$. Recall that the Kullback–Leibler divergence (Cover and Thomas 2006) between two probability distributions $p$ and $q$ is defined as

$$KL(p \,\|\, q) = \sum_x p(x) \log \frac{p(x)}{q(x)},$$

is nonnegative, and is zero if and only if $p = q$.

On the other hand, if $c$ is extremely small, say some $\epsilon > 0$, adding data will wildly change the code table to extremes, leading to larger codes for certain values, especially when the data is not very large. In both cases, for $|D| \to \infty$, the relative overhead converges to zero, however, in practice the amount of available data is limited, and therefore $c$ does have an impact on the encoded length. A natural and often chosen value is $c = 0.5$ (Grünwald 2007).

Let us consider the description length of the data for a single attribute cluster, using prequential coding. Let $l$ be the number of distinct values that can occur, let $m_i$ be the count of each such value $v_i$ in the domain of the cluster (i.e., $m_i = supp(v_i)$), and let $m = \sum_{i=1}^{l} m_i$ be the total number of transactions (i.e., $m = |D|$). If in the $j$th encountered tuple, we observe a value $v$ that has up till now occurred $m_v$ times, then its current frequency in the code table is $(m_v + 0.5)/(j + 0.5\,l)$. Hence its code length is $-\log((m_v + 0.5)/(j + 0.5\,l))$. Adding all code lengths together, we obtain

$$\begin{aligned} -\log \frac{\prod_{i=1}^{l} \prod_{j=0}^{m_i - 1} (j + 0.5)}{\prod_{j=0}^{m-1} (j + 0.5\,l)} &= \log \frac{\Gamma(m + 0.5\,l) / \Gamma(0.5\,l)}{\prod_{i=1}^{l} \Gamma(m_i + 0.5) / \Gamma(0.5)} \\ = \quad \log \Gamma(m + 0.5\,l) - \log \Gamma(0.5\,l) &- \sum_{i=1}^{l} \big( \log((2m_i - 1)!!) - m_i \big), \end{aligned}$$

where !! denotes the double factorial defined as $(2k-1)!! = \prod_{i=1}^{k} (2i-1)$, and $\Gamma$ is the gamma function, which is an extension of the factorial function to the complex plane, that is, $\Gamma(x+1) = x\Gamma(x)$, with relevant base cases $\Gamma(1) = 1$ and $\Gamma(0.5) = \sqrt{\pi}$.

Interestingly, even though the distribution and hence the code lengths constantly change during the encoding of $D$, the total description length does not depend on the order in which the tuples are processed.

The number of values $l$ in the formula above can be extremely large if we take it to be the size of the domain of a cluster. However, in practice it will be a lot smaller than the full domain, especially for large attribute sets. Therefore, we can first specify the values that can occur. We do this using a model-to-data code for all nonempty subsets of $dom(A)$, which takes $\log(2^{|dom(A)|} - 1)$ bits.

**Definition 5** The refined description length of the data for a single cluster $A$, using prequential coding, is given by

$$L_r(A, D) = \log(2^{|dom(A)|} - 1) + \log \Gamma(m + 0.5\,l) - \log \Gamma(0.5\,l) - \sum_{i=1}^{l} \big(\log((2m_i - 1)!!) - m_i\big),$$

where $l = |\{v \in dom(A) \mid fr(A = v) \neq 0\}|$ is the number of values with non-zero support in the data. The refined description length of an attribute clustering $\mathcal{C} = \{A_1, \ldots, A_k\}$, using prequential coding, is then equal to

$$L_r(\mathcal{C}, D) = \log(B_n) + \sum_{i=1}^{k} L_r(A_i, D).$$

### 3.8 Querying a summary

Besides providing insight into which attributes interact most strongly, and which of their values typically co-occur, our summaries can also be used as surrogates for the data. That is, they form probabilistic models, being the product of independent distributions on clusters of attributes that can easily be queried.

For categorical data, querying comes down to calculating marginal probabilities, i.e., determining itemset frequencies. The frequency of an itemset $(X = v)$ can simply be estimated from an attribute clustering by splitting up the itemset over the clusters, and calculating the product of the marginal frequencies of the subsets in each separate cluster.

This splitting up is justified by the fact that all relevant correlations between attributes are captured by a good clustering. Note that the more attributes are clustered together, the more detail the corresponding code table contains about their correlations, and hence allow for better frequency estimations. In fact, for the trivial complete clustering, where all attributes are grouped together, we will obtain exact frequencies. As our summaries are constructed with the goal of capturing the key correlations as well as possible using as few and simple code tables as possible, we expect to obtain highly accurate, but not exact frequency estimations.

**Definition 6** Let $\mathcal{C} = \{A_1, \ldots, A_k\}$ be a clustering of $\mathcal{A}$, and let $(X = v)$ be an itemset, with $v \in dom(X)$. Then the frequency of $(X = v)$ is estimated as

$$\hat{fr}(X = v) = \prod_{i=1}^{k} fr(X \cap A_i = v_i),$$

where $v_i$ is the sub-vector of $v$ for the corresponding attributes in $X \cap A_i$.

For instance, let $\mathcal{A} = \{a, b, c, d, e, f\}$ and $\mathcal{C} = \{abc, de, f\}$, and let us consider the itemset $(abef = 1)$, then $\hat{fr}(abef = 1) = fr(ab = 1) \cdot fr(e = 1) \cdot fr(f = 1)$.

Since the code table for each cluster $A_i$ contains the frequencies of the values for $A_i$—because of the one-to-one mapping between code length and frequency—we can use our clustering model as a very efficient surrogate for the database. For a cluster $A_i$, let $\Omega_i$ be the subset of values in $dom(A_i)$ having a non-zero frequency. The complexity of frequency estimation is then upper bounded by

$$O\left(\sum_{i=1}^{k} |A_i \cap X||\Omega_i|\right) \leq O\left(|X||D|\right).$$

If $|\Omega_i| \ll |D|$, querying the summary is significantly faster than querying the data. Note that this automatically holds for small clusters for which $|dom(A)| < |D|$.

## 4 Mining attribute clusterings

Now that we have defined how we can identify the best clustering, we need a way to *discover* it. In this section we present our algorithm, and investigate its properties and computational complexity.

### 4.1 Algorithm

As discussed in Sect. 3.4, the search space we have to consider is extremely large. Furthermore, our quality score is not (anti-)monotonic, excluding the possibility of efficient search. Hence, it is infeasible to examine the search space exhaustively, and thus we settle for heuristics. In this section we introduce our algorithm, which finds a good attribute clustering $\mathcal{C}$ for a dataset $D$, with a low description length $L(\mathcal{C}, D)$.

We use a greedy bottom-up hierarchical clustering algorithm that traverses the search space by iteratively merging clusters such that in each step the description length is minimized. The pseudo-code is given in Algorithm 1. We start by placing each attribute in its own cluster (line 1), which corresponds to the independence model. Then, we iteratively find the pair of clusters whose merger results in a clustering with the smallest description length. From Sect. 3.5 we know that this is the pair of clusters with the highest similarity, which can be computed locally (5). The clusters are merged (6), and the algorithm continues. We maintain the clustering with the shortest description (8–9), and finally return the best clustering (10).

**Algorithm 1:** ATTRIBUTECLUSTERING($D$)

**input** : Categorical dataset $D$ over a set of attributes $\mathcal{A}$.
**output** : Attribute clustering $\mathcal{C} = \{A_1, \ldots, A_k\}$.

$\mathcal{C} \leftarrow \{\{a\} \mid a \in \mathcal{A}\}$
$\mathcal{C}_{\min} \leftarrow \mathcal{C}$
Compute and store $CS_D(A_i, A_j)$ for all $i \neq j$
**while** $|\mathcal{C}| > 1$ **do**
  $A_i, A_j \leftarrow \arg\max_{i,j} CS_D(A_i, A_j)$
  $\mathcal{C} \leftarrow \mathcal{C} \setminus \{A_i, A_j\} \bigcup \{A_i \cup A_j\}$
  Compute and store $CS_D(A_{ij}, A_l)$ for all $l \neq ij$
  **if** $L(\mathcal{C}, D) < L(\mathcal{C}_{\min}, D)$ **then**
    $\mathcal{C}_{\min} \leftarrow \mathcal{C}$
**return** $\mathcal{C}_{\min}$

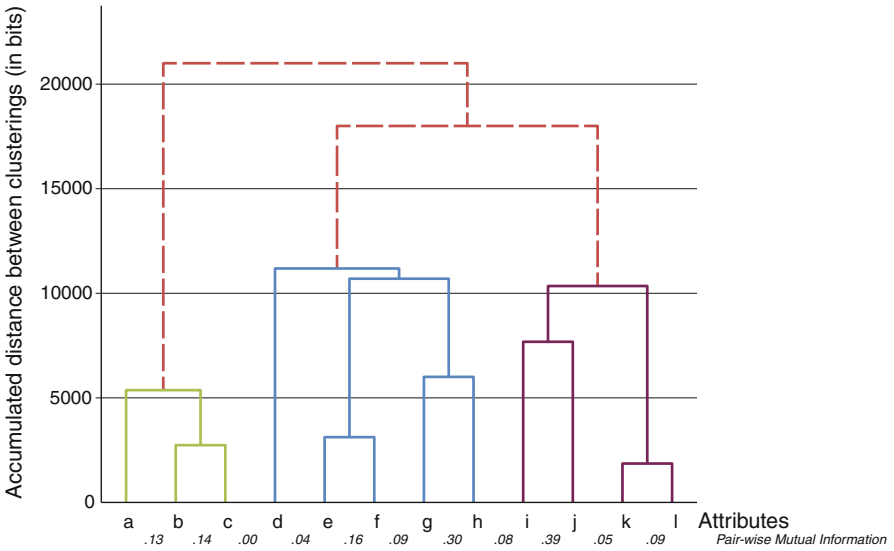


**Fig. 2** Example of a dendrogram depicting a hierarchy of attribute clusterings, for a subset of the attributes of the categorical *Markov* dataset. Merges that save bits are depicted with *solid lines*; their height corresponds to the cumulative drop in description length. There are three attribute clusters in this example

This results in a hierarchy of clusters, which can be represented visually as a dendrogram, as shown in Fig. 2. The clustering at the bottom corresponds to the independence distribution, while the clustering at the top represents the joint empirical distribution of the data. In the figure, merges which result in a lower description length are depicted in solid lines. Their height corresponds to the (cumulative) decrease in description length. An advantage of this approach is that it allows us to visualize how the clusters were formed, and how they are structured internally.

The graph in Fig. 3 shows how the description length behaves as a function of $k$ (the number of clusters), during a run of the algorithm on the binary *Connect* dataset (Frank and Asuncion 2010) (see Sect. 7 for more details on this dataset). Starting at $k = n$,

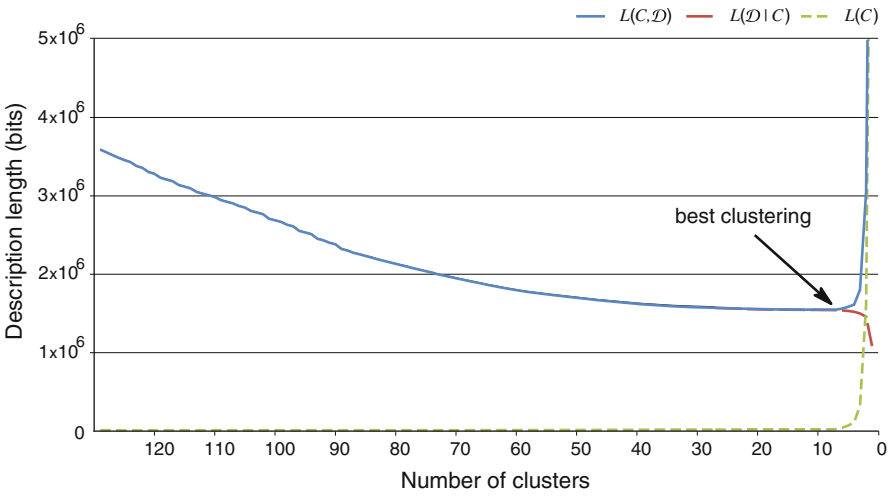


**Fig. 3** Evolution of the encoded length $L(\mathcal{C}, D)$ with respect to the number of clusters $k$, on the binary *Connect* dataset (Frank and Asuncion 2010). The best clustering is located at $k = 7$

the description length $L(\mathcal{C}, D)$ gradually decreases as correlated clusters are merged. This indicates that there is structure present in the data, which is exploited to obtain a shorter description of it. Note that the description length of the data, $L(D \mid \mathcal{C})$, decreases monotonically, since a less refined clustering uses more information, and hence fits the data better. The total description length continues to decrease until $k = 7$, which yields the best clustering found for this dataset. Afterwards, $L(\mathcal{C}, D)$ increases again, due to the fact that the decrease in $L(D \mid \mathcal{C})$ does not outweigh the dramatic increase of $L(\mathcal{C})$, which means that the models are getting far too complex past this point.

## 4.2 Stopping criterion

Figure 3 seems to suggest that $L(\mathcal{C}, D)$, as a function of $k$, has a single minimum. That is, first the description length decreases until a local optimum is reached, and afterwards the description length only increases. Naturally, the question arises whether this is the case in general; if so, the algorithm can terminate as soon as a local minimum is detected. Intuitively, we would expect that if the current best cluster merger increases the total description length, then any future merger is probably also a bad one. However, the following counterexample shows that this is not necessarily true.

Consider the dataset $D$ in Table 2 with four binary attributes, $\mathcal{A} = \{a, b, c, d\}$. Assume that $a$, $b$, and $c$ are independent, and that for every transaction of $D$ it holds that $d = a \oplus b \oplus c$, where $\oplus$ denotes exclusive or (XOR). Now, using this dependency, let $D$ contain a transaction for every $v \in \{0, 1\}^3$ as values for $abc$, with multiplicity, say, 10. Then every pair of clusters whose union contains strictly less than four attributes (e.g., $A_i = ab$ and $A_j = d$) is independent. As the algorithm starts to merge clusters, the data description length $L(D \mid \mathcal{C})$ remains constant (namely $4|D|$), but

**Table 2** An exclusive or (XOR) dataset

| $a$ | $b$ | $c$ | $d$ | Multiplicity |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | × 10 |
| 0 | 0 | 1 | 1 | × 10 |
| 0 | 1 | 0 | 1 | × 10 |
| 0 | 1 | 1 | 0 | × 10 |
| 1 | 0 | 0 | 1 | × 10 |
| 1 | 0 | 1 | 0 | × 10 |
| 1 | 1 | 0 | 0 | × 10 |
| 1 | 1 | 1 | 1 | × 10 |

The last attribute equals the XOR of the first three. The rightmost column denotes the number of times each transaction occurs, i.e., there are 80 transactions

the code tables become more complex, and thus $L(\mathcal{C})$ and $L(\mathcal{C}, D)$ increase. Only at the last step, when the two last clusters are merged, can the structure be captured and $L(D \mid \mathcal{C})$ decreases to $3|D|$, leading to a decrease of the total description length.

Note, however, that the drop in encoded length in the last step depends on the number of transactions in the database. In the above example, if the multiplicity of every unique transaction is 10, the complete clustering would be preferred over the independence clustering. However, if there are fewer transactions (say, every unique transaction occurs only once), then even though the dependencies are the same, the algorithm decides that the best clustering corresponds to the independence model, i.e., that there is no significant structure. Intuitively this can be explained by the fact that if there is only a small number of samples then the observed dependencies might be coincidental, but if many transactions exhibit it, the dependencies are truly present. This is one of the benefits we get from using MDL to select models.

This example shows that in general we should not stop the algorithm when the description length attains a local minimum. Nevertheless, it is a very synthetic example with a strong requirement on the number of transactions. For instance, if we generalize the XOR example to 20 attributes, the minimum number of transactions for it to be detectable already runs in the millions. Moreover, in none of our experiments with real data did we encounter a local minimum that was not also a minimum over all considered clusterings. Therefore, we can say that for all practical purposes it is acceptable to stop the algorithm when we encounter a clustering with a locally minimal description length.

### 4.3 Algorithmic complexity

Naturally, a summarization method should be fast, because our aim is to get a quick overview of the data. Notwithstanding the search for efficient algorithms in data mining, in practice many exhibit an exponential runtime. Here we show that our algorithm is polynomial in the number of attributes.

In the first iteration, we compute the description length of each singleton $\{a\}$, with $a \in \mathcal{A}$, and then determine which two clusters to merge. To do this, we must compute $O(n^2)$ cluster similarities, where $n = |\mathcal{A}|$. Since we might need some of these similarities later on, we store them in a heap, such that we can easily retrieve the maximum. Now say that in a subsequent iteration $k$ we have just merged clusters $A_i$ and $A_j$ into $A_{ij}$. Then we can delete the $2k - 3$ similarities from the heap that are no longer of

use, and we need to compute and insert $k-2$ new similarities, namely those between $A_{ij}$ and the remaining clusters. Since heap insertion and deletion is logarithmic in the size of a heap, maintaining the similarities in an iteration with $k$ clusters, takes $O(k \log k)$ time. The initial filling of the heap takes $O(n^2 \log n)$. Since the algorithm performs at most $N$ iterations, we see that the time complexity of maintaining the cluster similarities is $O(n^2 \log n) + \sum_{k=n}^{1} O(k \log k) = O(n^2 \log n)$.

Next we discuss the complexity of computing the similarities. Say that in an iteration $k$, we must compute the similarities $CS_D(A_i, A_j)$ between the last merged cluster $A_i$ and all other remaining clusters $A_j$. This requires collecting all non-zero frequencies $fr(A_{ij} = v)$, and we do this by simply iterating over all transactions $t$ and computing $|A_{ij} \cap t|$, which takes $O(n|D|)$ per pair in the worst case. Computing the initial similarities between all pairs of singletons takes $O(n^2|D|)$. Hence, over all iterations, the similarity computations take $O(n^2|D|) + \sum_{k=n}^{1} O(kn|D|) = O(n^3|D|)$.

Therefore, the total time complexity of AttributeClustering is

$$O(n^3|D|).$$

The dominant cost in terms of storage comes from the heap containing the cluster similarities, and hence the memory complexity of our algorithm is $O(n^2)$.

## 5 Alternative approaches

Since we employ a greedy algorithm, we do not necessarily find the globally optimal attribute clustering. Here we discuss some potential alternative search strategies.

### 5.1 Vertical clustering

A very straightforward approach to clustering attributes is to use any one of the wealth of existing clustering algorithms that cluster rows of data, and adapt it to our setting. By simply transposing the data, and applying say, $k$-means, we are done. However, the issue is not that trivial. While transposing data could conceptually make sense for binary data, this is arguably not the case for categorical data in general. Even so, most, if not all, clustering algorithms require a distance measure between tuples, which means we would need a distance measure for attributes. For categorical data, an often used measure is Hamming distance, the number of locations with unequal values. If two attributes have the same value in many transactions, clearly they are very similar. However, the converse is not true, e.g., two binary attributes that are each other's inverse are clearly correlated. Moreover, attributes with different domains would be incomparable. A similar argument can be made for Manhattan or Euclidean distances.

Turning to the area of information theory, a good choice is to use a distance measure based on mutual information. Specifically, let $a_1$ and $a_2$ be two attributes, then we could use the distance $d(a_1, a_2) = H(a_1, a_2) - I(a_1, a_2)$. This is a proper metric: it is symmetric, nonnegative, zero if and only if both attributes are a function of one another (i.e., there exists a bijection between them), and it obeys the triangle

inequality. It reaches a maximum of $H(a_1, a_2)$ when $a_1$ and $a_2$ are independent. We could additionally choose to normalize $d$, e.g., by dividing by $H(a_1, a_2)$.

However, using pairwise distances on attributes can only get us so far, as the following small example shows. Consider a dataset with six binary attributes that is the Cartesian product of two (independent) XOR truth tables. Each pair of attributes in this dataset is independent, and has the same distance of 2. Consequently, no algorithm based on pairwise distances will be able to detect that there are two groups of three correlated attributes in this dataset.

Finally, we might consider directly using the mutual information between clusters. Let $A_1$ and $A_2$ be two attribute clusters, then the distance between the clusters is $d(A_1, A_2) = H(A_1, A_2) - I(A_1, A_2)$. In this case, the best clustering is always the complete clustering. In the absence of a complexity penalty term, this requires the user to specify a parameter $k$ for the number of clusters. Moreover, the choice of a specific search strategy is crucial here. The mutual information between a large and a small cluster will typically be larger than the mutual information between two small clusters. If one cluster becomes large, it will typically have a high similarity with other clusters. Hence, it will tend to 'eat up', as it were, the smaller clusters, rather than that the small clusters would merge together, resulting in a very heterogeneous, unbalanced clustering, which is not likely to be a desirable result.

### 5.2 Divisive hierarchical clustering

Instead of taking a bottom-up, agglomerative approach, a top-down divisive hierarchical clustering algorithm could be considered. Such an algorithm would start at the least refined clustering, and then iteratively split a cluster such that the description length decreases maximally. However, this approach is far from feasible, since already in the first step, the algorithm needs to consider $2^{|\mathcal{A}|-1} - 1$ possible splits (i.e., there are $2^{|\mathcal{A}|} - 2$ proper nonempty subsets; their complements describe the same splits). While a divisive approach does consider more clusterings than an agglomerative one, and hence could possibly result in a clustering with a lower description length, its exponential behavior makes it unsuitable for summarization purposes.

### 5.3 Beam search

Beam search attempts to balance the level-wise, large-coverage approach of breadth-first search, with the memory-efficiency of depth-first or greedy search. Starting with an initial state, a *beam* containing the $b$ best solutions is maintained at each level, where $b$ is called the beam width. The beam at the next level consists of the $b$ best scoring neighbor states of the solutions in the current beam. Note that when the beam width is set to one, beam search is equivalent to our greedy algorithm. On the other hand, if we set $b$ to infinity, the algorithm is simply breadth-first search. Since beam search considers a larger part of the search space, it generally performs better for larger values of $b$. However, in practice the beam often exhibits limited variability, and may hence not lead to considerably better results. Further, both time and memory

complexity of the algorithm are multiplied by a factor $b$. We will empirically compare our approach to beam search in Sect. 7.

### 5.4 Simulated annealing

Simulated annealing (Kirkpatrick 1984) is a probabilistic optimization technique that tries to avoid finding a locally but not globally optimal solution, by not only greedily moving toward better neighbor solutions in the search space, but also heuristically moving to worse ones. The idea is borrowed from the physical process of annealing, where a material is heated and then gradually cooled down to form a crystal structure, which is a low energy state. The probability of transitioning to a neighboring state is dependent on the difference in energy of both states, and the current temperature of the system. As the temperature decreases, the probability of going to a higher energy state decreases. Many parameters influence the end result, such as the initial temperature, the cooling schedule, the material, etc.

In our setting, states are attribute clusterings, and the neighbors of a clustering $\mathcal{C}$ are all clusterings that are the result of either merging two of its clusters, or splitting one. The description length acts as the energy function. For the temperature, we employ a geometric cooling schedule: in iteration $i$, the temperature is decreased by a factor $\alpha : t_{i+1} = \alpha t_i$. In this way, the number of iterations $m$ and the cool down rate are directly linked. We choose $t_0 = 1$, and set $\alpha$ such that $t_m = \alpha^m$ becomes equal to zero in double floating point precision. When in a certain state $\mathcal{C}$, we randomly choose to do either a split or a merge, and then uniformly pick a neighbor state $\mathcal{C}'$, and compute the difference in description length. The energy is normalized by estimating the maximal possible difference by sampling.

The probability of switching from state $\mathcal{C}$ to state $\mathcal{C}'$ in iteration $i$ is given by

$$\Pr(\mathcal{C} \to \mathcal{C}') = \exp \frac{-CS(\mathcal{C}, \mathcal{C}')/N}{\lambda \alpha^i}.$$

The $\lambda$ constant acts as a scaling parameter. If $CS$ is positive, i.e., if $\mathcal{C}'$ has a lower description length than $\mathcal{C}$, the transition is always made. On the other hand, if $CS$ is negative, $\mathcal{C}'$ has a higher description length, and the transition is only made probabilistically based on the magnitude of $CS$ and the current temperature, which decreases over time. The algorithm is initiated in a random clustering.

Since simulated annealing has the potential to move away from a local minimum, unlike true greedy approaches, it has the potential to find better solutions. However, due to its inherently non-deterministic nature, a single run is not likely to give us a satisfactory answer—rather, the algorithm will have to be run many times, and even then we still have no guarantee that the best solution we encountered is anywhere close to the global optimum. Furthermore, by considering both groupings and splits of the current clustering, this approach is computationally heavy, as we have to calculate up to an exponential number of switching probabilities per step. We will empirically compare our greedy agglomerative approach to simulated annealing in Sect. 7.

## 6 Related work

The main goal of our proposed method is to offer a good first impression of a categorical dataset. For numerical data, averages and correlations can easily be computed, and more importantly, are informative. For binary and categorical data, such informative statistics beyond simple counts are not readily available. As such, our work can be seen as to provide an informative 'average' for categorical data; for those attributes that interact strongly, it shows how often their value combinations occur.

Many existing techniques for summarization are aimed at giving a succinct representation of a given collection of patterns. Well-known examples include closed itemsets (Pasquier et al. 1999) and non-derivable itemsets (Calders and Goethals 2007), which both provide a lossless reduction of the complete pattern collection.

A lossy approach that provides a succinct summary of the patterns was proposed by Yan et al. (2005). The authors cluster groups of itemsets and describe these groups using *profiles*, which can be characterized as conditional independence distributions on a subset of the items, which can subsequently be queried. Experiments show that our method provides better frequency estimates, while requiring fewer clusters than profiles. Wang and Karypis (2004) give a method for directly mining a summary of the frequent pattern collection for a given *minsup* threshold. Han et al. (2007) provide a more complete overview of pattern mining and summarization techniques.

For directly summarizing data rather than pattern sets much fewer proposals exist. Chandola and Kumar (2005) propose to induce $k$ transaction templates such that the database can be reconstructed with minimal loss of information. Alternatively, the KRIMP algorithm (Vreeken et al. 2011; Siebes et al. 2006) selects those itemsets that provide the best lossless compression of the database, i.e., the best description. While it only considers the 1s in the data, it provides high-quality and detailed results, which are consequently not as small and easily interpreted as our summaries. Though the KRIMP code tables can generate data virtually indistinguishable from the original (Vreeken et al. 2007), they are not probabilistic models and cannot be queried directly, so they are not immediately suitable as data surrogates.

Wang and Parthasarathy (2006) build probabilistic Maximum Entropy models of the data by incrementally adding those itemsets into the model that deviate more than a given error threshold—considering the data as a bag of independent samples. The approach ranks and adds itemsets in level-wise batches, i.e., first itemsets of size 1, then of size 2, and so on. Mampaey et al. (2011) improve over this method by avoiding the level-wise approach, and instead iteratively incorporate the itemset that increases the likelihood of the model most. Furthermore, by employing the Bayesian Information Criterion, the set of most informative itemsets can be identified without requiring any parameters. De Bie (2011) proposes to use the Maximum Entropy principle to instead model the data as a whole—considering it as a monolithic sample. In (Kontonasios and De Bie 2010) he shows this model can be used very effectively to rank and select itemsets with regard to the information they provide, while also taking their complexity into account.

All the above-mentioned techniques differ from our approach in that they model the data in relatively high detail using itemsets, whereas we provide a more high level

summary, that identifies the most strongly interacting categoric attributes, and their most prevalent attribute-value combinations.

Somewhat related to our method are low-entropy sets (Heikinheimo et al. 2007), attribute sets for which the entropy lies below a given threshold. As entropy is strongly monotonically increasing, typically very many low-entropy sets are discovered even for low thresholds, and hence, this pattern explosion is often even worse than in frequent itemset mining. To this end, Heikinheimo et al. (2009) introduced LESS, a filtering proposal called to select those low-entropy sets that together describe the data well. In our approach, instead of filtering, we discover attribute sets with low entropy directly on the data.

Orthogonal to our approach, the maximally informative $k$-itemsets (miki's) by Knobbe and Ho (2006) are $k$ items (or patterns) that together split the data optimally, found through exhaustive search. Bringmann and Zimmermann (2007) propose a greedy alternative to this exhaustive method that can consider larger sets of items. Our approach groups attributes together that correlate strongly, so the correlations between groups are weak. As future work, we plan to investigate whether good approximate miki's can be extracted from our summaries.

Since our approach employs clustering, the work in this field is not unrelated. However, clustering is foremost concerned with grouping rows together, typically requiring a distance measure between objects. Co-clustering (Chakrabarti et al. 2004), or bi-clustering (Pensa et al. 2005), is a type of clustering in which clusters are simultaneously detected over both attributes and rows. Chakrabarti et al. (2004) also employ the MDL principle to identify the best clustering, however, whereas our approach is to identify groups of categoric attributes for which the attribute-value combinations show strong correlation, their approach identifies locally dense areas in sparse binary matrices, and is not trivially extendable for categoric data.

Au et al. (2005) present an algorithm that clusters features in gene expression data, taking into account a target attribute, in order to do classification. To this end the authors introduce a distance measure between attributes, based on their mutual information. However, as argued in Sect. 5, this may not always be a good choice. The algorithm is a variant of the $k$-means algorithm—using mode attributes rather than means. Since the number of clusters $k$ is a parameter of the algorithm, the authors propose to simply run the algorithm for all possible values of $k$, and select the clustering minimizing a defined score. The application of the paper is classification of gene expression data, which often suffers from the $d \ll n$ problem, i.e., the number of attributes ($n$) is far greater than the number of samples ($d$). This has implications for the significance of the correlation (or mutual information) between attributes, and might lead to overfitting. Our algorithm takes this into account by using MDL.

Dhillon et al. (2003) provide an algorithm that clusters words in text data for classification. For each word cluster, a feature is constructed as a weighted average of the distributions of the separate words in the cluster. While this inherently discards some information, it also reduces the number of features, making classification easier, and does not discard as much information as feature selection would. To find the clusters, an algorithm similar to $k$-means is presented. The word clusters are then used in a naive Bayes classifier. The number of clusters $k$ is a parameter of the algorithm, however, in this case it arguably is not a goal to find an optimal clustering, but to reduce the

number of features purely for performance reasons, and hence it is probably desirable to be able to control this parameter.

Our formalization can also be regarded as a distance measure between categoric attributes (or, categoric data in general). As such, the proposal by Das et al. (1997) is both interesting and relevant. There, the authors take an orthogonal approach by measuring the similarity of a set of binary attributes not by regarding the similarity over the selected attributes, but by considering the marginal frequencies of a set of *other* attributes, called probes. Although experimental results show that some true similarities between attributes are captured, the measure and its results do lack an intuitive interpretation, and the selection of probes is manual, requiring further development in order to be used in practice.

## 7 Experiments

In this section we experimentally evaluate our method and validate the quality of the discovered attribute clusterings.

### 7.1 Setup

We implemented our algorithm in C++, and make the source code available for research purposes.[1] All experiments were executed on 2.67 GHz (six-core) Intel Xeon X5650 machines with 12 GB of memory, running Linux, Ubuntu 11.4. All reported runtimes were obtained using a single-threaded version of the implementation. Unless stated otherwise, empirical $p$-values were calculated against the scores of 1,000 randomized models or datasets, providing a resolution of at most 0.1%. For all experiments we recorded memory usage during summary construction, which was at most a few megabytes excluding the database.

### 7.2 Datasets

We evaluate our method on nineteen different datasets, covering a wide range of different data characteristics. We use five synthetic datasets and 14 real-world and benchmark datasets, all of which, save one, are publicly available. Their basic characteristics are depicted in Table 3.

The binary and categorical *Independent* datasets have independent attributes with randomly drawn frequencies. The attributes of the *Markov* datasets form a Markov chain. In the binary version each attribute is a copy of the previous one with a random copy probability, the first attribute having a 50% probability of being one. Similarly, the categorical version has attributes with up to eight values per attribute, and each attribute depends on the previous one according to a randomly generated contingency table. The categorical *Independent* and *Markov* data are generated such that they have the same column margins. The *DAG* dataset is generated according to a directed

[1] http://www.adrem.ua.ac.be/implementations.

**Table 3** The basic characteristics of the datasets used in the experiments

| | $\|\mathcal{A}\|$ | $\|D\|$ | $L(\mathcal{I}, D)$ | $L_c(D)$ |
|---|---|---|---|---|
| Binary data | | | | |
| Independent | 50 | 20,000 | 895,079 | 1,000,000 |
| Markov | 50 | 20,000 | 999,159 | 1,000,000 |
| DAG | 50 | 20,000 | 970,485 | 1,000,000 |
| Accidents | 468 | 340,183 | 25,991,622 | 159,205,644 |
| BMS-Webview-1 | 497 | 59,602 | 1,201,522 | 29,622,194 |
| Chess | 75 | 3,196 | 142,812 | 239,700 |
| Connect | 129 | 67,557 | 3,593,260 | 8,714,853 |
| DNA Amplification | 391 | 4,590 | 191,428 | 1,794,690 |
| Mammals | 121 | 2,183 | 121,572 | 264,143 |
| MCADD | 198 | 31,924 | 2,844,465 | 6,320,952 |
| Mushroom | 119 | 8,124 | 443,247 | 966,756 |
| Pen Digits | 86 | 10,992 | 605,413 | 945,312 |
| Categorical data | | | | |
| Independent | 50 | 20,000 | 2,037,016 | 2,109,825 |
| Markov | 50 | 20,000 | 2,036,871 | 2,109,824 |
| Chess | 37 | 3,196 | 71,651 | 120,122 |
| Connect | 43 | 67,557 | 2,013,066 | 4,604,239 |
| MCADD | 22 | 31,924 | 1,966,658 | 2,168,968 |
| Mushroom | 23 | 8,124 | 267,334 | 388,268 |
| Pen Digits | 17 | 10,992 | 377,801 | 430,723 |

Shown are the number of attributes $|\mathcal{A}|$, the number of transactions $|D|$, the description length of the independence clustering $L(\mathcal{I}, D)$, and the canonical description length $L_c(D)$, both in bits

acyclic graph among its binary attributes. Each attribute depends on at most four of its preceding attributes, according to randomly generated contingency tables.

Next, we use 14 real-world and benchmark datasets. The well-known categorical *Chess*, *Connect*, and *Mushroom* datasets were obtained from the UCI Machine Learning Repository (Frank and Asuncion 2010). Their binary counterparts were obtained from the FIMI Dataset Repository (Goethals and Zaki 2003), and simply contain one binary attribute for each attribute-value pair in the categorical versions.

The *Accidents* and *BMS-Webview-1* datasets were also obtained from the FIMI Dataset Repository.

The *DNA Amplification* database contains data on DNA copy number amplifications. Such copies activate oncogenes and are hallmarks of nearly all advanced tumors (Myllykangas et al. 2006). Amplified genes represent targets for therapy, diagnostics and prognostics.

The *Mammals* data[2] consists of presence records of European mammals within geographical areas of $50 \times 50\,\text{km}^2$ (Mitchell-Jones et al. 1999).

[2] http://www.european-mammals.org/.

The categoric *MCADD* data was obtained from the Antwerp University Hospital. Medium-Chain Acyl-coenzyme A Dehydrogenase Deficiency (MCADD) (Baumgartner et al. 2005; Van den Bulcke et al. 2011) is a deficiency newborn babies are screened for during a Guthrie test on a heel prick blood sample. The instances are represented by a set of 21 features: 12 different acylcarnitine concentrations measured by tandem mass spectrometry (TMS), together with 4 of their calculated ratios and 5 other biochemical parameters, each of which we discretized using $k$-means clustering with a maximum of 10 clusters per feature.

Finally, the *Pen Digits* data was obtained from the LUCS-KDD data library (Coenen 2003), and contains handwritten samples of the digits 0–9. The attributes correspond to eight $x$ and $y$ coordinates on a grid, describing the trace of a certain digit, which is indicated by the class label.

### 7.3 Evaluation

Table 4 presents an overview of the results of our algorithm for the used datasets. We show the number of clusters $k$ in the clustering that our algorithm finds, its description length $L(\mathcal{C}, D)$—both absolute and relative to the both the description length of the independence clustering $L(\mathcal{I}, D)$ and the canonical description length $L_c(D)$—and the wall clock time the algorithm took to complete. A low number of clusters and a short description length indicate that our algorithm models structure that is present in the data. For most datasets we see that the number of clusters $k$ is much lower than the number of attributes $|\mathcal{A}|$. In fact, these numbers are such that it is indeed feasible to inspect these clusters by hand. Many of the datasets are highly structured, which can be seen from the strong compression ratios the clusterings achieve with respect to their canonical description lengths. Lastly, the table also shows that our algorithm usually needs just a handful of seconds to complete.

Below, we investigate the clusterings discovered by our algorithm in closer detail.

For the categorical *Independent* data, we see that the algorithm identifies 50 clusters of single attributes, which correctly corresponds to the generating independence distribution. For the binary *Independent* data, though, the algorithm concludes that there are 49 clusters. Upon further inspection, it turns out that the two attributes that are grouped together are not really independent in the data. In fact, their joint distribution differs significantly from their product distribution ($p$-value 0.04). This actually makes an interesting point, because MDL does not specifically attempt to discover the 'true' underlying model (which for the synthetic data is available to us), but instead simply tries to describe the data best. For the data instance under consideration here, where by chance two attributes happen to be somewhat correlated, the algorithm cannot but conclude that they should be clustered together.

The attributes in both *Markov* datasets form a Markov chain, so we expect that nearby attributes are clustered together. This is indeed what happens: each cluster consists of consecutive attributes. Figure 2 shows the resulting dendrogram for a subset of the attributes in the categorical dataset. Further, if we look at the mutual information between pairs of adjacent attributes, the pairs with high mutual information tend to be grouped together, whereas pairs with low mutual information are not.

**Table 4** Results of our attribute clustering algorithm

| | $k$ | $L(\mathcal{C}, D)$ | Time | Compression ratio (%) | |
|---|---|---|---|---|---|
| | | | | $L(\mathcal{I}, D)$ | $L_c(D)$ |
| Binary data | | | | | |
| Independent | 49 | 895,078 | 1 s | 99.9 | 89.5 |
| Markov | 15 | 884,943 | 4 s | 88.6 | 88.5 |
| DAG | 12 | 797,588 | 5 s | 82.2 | 79.8 |
| Accidents | 116 | 16,893,671 | 120 min | 65.0 | 10.6 |
| BMS-Webview-1 | 140 | 1,078,905 | 16 min | 89.8 | 3.6 |
| Chess | 11 | 58,457 | 1 s | 40.9 | 24.4 |
| Connect | 7 | 1,559,773 | 88 s | 43.4 | 17.9 |
| DNA Amplification | 56 | 82,209 | 33 s | 42.9 | 4.6 |
| Mammals | 28 | 98,515 | 2 s | 81.0 | 37.3 |
| MCADD | 12 | 1,816,628 | 146 s | 63.9 | 28.7 |
| Mushroom | 9 | 169,425 | 13 s | 38.2 | 17.5 |
| Pen Digits | 5 | 333,032 | 10 s | 55.0 | 35.2 |
| Categorical data | | | | | |
| Independent | 50 | 2,037,016 | 4 s | 100.0 | 96.5 |
| Markov | 20 | 1,932,611 | 12 s | 94.9 | 91.6 |
| Chess | 9 | 57,353 | 1 s | 80.0 | 47.7 |
| Connect | 7 | 1,554,827 | 11 s | 77.2 | 33.8 |
| MCADD | 11 | 1,785,850 | 6 s | 90.8 | 82.3 |
| Mushroom | 3 | 150,012 | 1 s | 56.1 | 38.6 |
| Pen Digits | 5 | 309,788 | 1 s | 82.0 | 71.9 |

Shown are the number of identified groups of attributes $k$, the description length $L(\mathcal{C}, D)$ and the wall clock time used to discover the clusterings. Further, we show the relative description lengths $\frac{L(\mathcal{C},D)}{L(\mathcal{I},D)}$ and $\frac{L(\mathcal{C},D)}{L_c(D)}$ with respect to the description length of the independence clustering and canonical description length

Likewise, in the *DAG* dataset, which has attribute dependencies forming a directed acyclic graph, the clusters contain attributes which form tightly linked subgraphs. Figure 4 depicts the dependency graph between the attributes of the *DAG* dataset. The width of the edges is proportional to the mutual information between the corresponding attributes. Note that this is done for illustration purposes only, since pairwise mutual information does not capture all information, as pointed out in Sect. 5. The discovered clusters are drawn in rectangles. As the figure shows, the edges inside clusters are generally bold, while the few inter-cluster edges are usually thin. This is a good indication of the quality of the discovered clustering.

The *DNA Amplification* dataset is an approximately *banded* dataset (Garriga et al. 2011): the majority of the ones form a staircase pattern, and are located in blocks along the diagonal. In Fig. 5, a selected subset of the rows and columns of the data is plotted, along with some of the attribute clusters that our algorithm finds. The 1s in the data are drawn dark, the 0s are white. For presentation purposes the figure has been rotated. The clustering clearly distinguishes the blocks that form the staircase pattern, even though

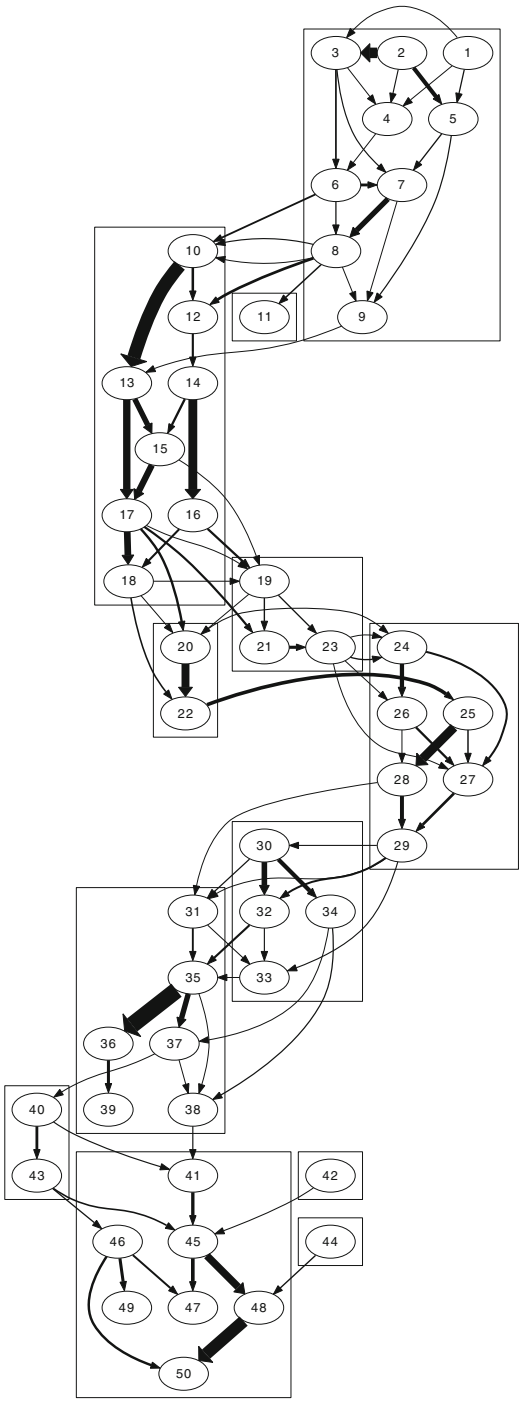


**Fig. 4** Dependency graph of the attributes of the *DAG* dataset, with the discovered attribute clusters drawn in *rectangles*. The width of an edge between two nodes is proportional to the mutual information between the corresponding attributes

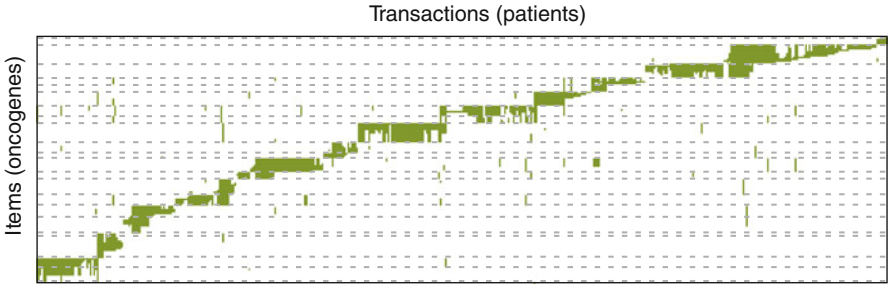


**Fig. 5** A selected sub-matrix of the *DNA Amplification* data and the corresponding discovered attribute clusters, which are separated by the *dotted lines*. For presentation purposes the figure is rotated sideways

the data is quite noisy. These blocks correspond to related oncogenes that are often amplified in conjunction. Further, the clusters contain genes that are located close to each other on their respective chromosomes. Inspecting the code tables, we see that the attributes within them are strongly correlated: for all code tables, the value with the highest frequency consists of zeroes (well over 90%, note this is a sparse dataset), and is followed by the value consisting of ones, whereas the remaining values have considerably lower frequencies. This shows that the attributes in a single are usually either all present or all absent, far more than expected under independence.

The *Connect* dataset contains all legal grid configurations with eight discs, of the well-known Connect Four game. The game grid has 7 columns and 6 rows, and for each of the 42 locations there is an attribute describing whether it is empty, or which one of the two players has positioned a disc there. Furthermore, a class label describes which player can win or whether the game will result in a draw—note that as we are mining the data exploratory, we consider the class label as 'just another attribute'. For both versions of the data, the algorithm discovers seven clusters. Upon inspection, it turns out that each of these clusters corresponds to a single column in the game, i.e., the structure found by the algorithm reflects the physical structure of the game, even though our algorithm has no knowledge of this. Whereas different columns are more or less independent, attributes within a column are dependent: an empty location cannot have a nonempty location above it. This is also observed in the code tables: all occurring values correspond to configurations with zero or more filled locations at the bottom, with empty locations above that. Since the full grid contains only eight discs, the highest frequency values in the code tables are the relatively sparse ones. Furthermore, the class label is placed in the cluster of the middle column; this is a very plausible finding, since any horizontal or diagonal row must necessarily pass through the middle column, making it key to winning the game. Additionally, we note that in the binary dataset items originating from the same categorical attribute are grouped together, rather than being spread over different clusters.

For both the *Chess* and *Mushroom* datasets, we observe that the discovered clusterings for the categorical and binary datasets are not exactly the same. The algorithm makes a few more merges in the categorical case, but otherwise the discovered clusterings are comparable. For these datasets we also see in the binary variants that items originating from the same attribute in general tend to be clustered together.

Interestingly, the (absolute) description lengths are very similar for both variants of the datasets, being slightly smaller in the categorical case. The explanation lies in the fact that for the categorical data we already know which values a given attribute may assume, while for the binary versions we do not know which items belong to the same attributes, which indirectly increases the cost to describe the code tables.

The attributes of the *MCADD* dataset consist of 12 different acylcarnitine concentrations measured by TMS, together with 4 of their calculated ratios and 5 other biochemical parameters, and the class label which indicates whether MCADD is present. This recessive metabolic disease affects about one in 10,000 people while around one in 65 is a carrier of the responsible mutated gene. If left undiagnosed, this rare disease is fatal in 20–25% of the cases and many survivors are left with severe brain damage after a severe crisis. In the results, we see that acylcarnitines and corresponding calculated ratios are detected and grouped together. For instance, the attributes for $C8$ and $C10$ are in a cluster together with the ratio $\frac{C8}{C10}$. Further, the class label cluster also contains the ratio $\frac{C8}{C12}$, which is one of the features commonly used in diagnostic criteria by experts and was also discovered in previous in-depth studies (Baumgartner et al. 2005).

Finally, for the *Pen Digits* datasets, we discover five attribute clusters. Each cluster consists entirely of either $x$ or $y$ coordinates. Besides spatially, the coordinate attributes are also temporally clustered: coordinates appearing first along the trace of a digit are grouped together, as are coordinates drawn later. More specifically, the $x$ coordinates are split in three groups: beginning, middle, and end. The $y$ coordinates are split in a beginning, and a middle-end cluster. This last cluster also contains the class label, which means that the label is correlated to the vertical coordinates of the pen when drawing the latter half of a digit, a statement the authors deem plausible. The binary variant of this datasets results in a very similar summary, i.e., barring a few exceptions, items originating from the same attributes are grouped together in the same way as with the categorical data.

The above examples show that the attribute clusterings discovered by our algorithm are of high quality. We find structure between correlated attributes, which can be seen from the strong compression ratios and relatively small number of clusters. When subjectively investigating the resulting clusters themselves, they are easily interpretable and highly insightful.

Having established that we achieve good results, we investigate the contribution our encoding. We compare our formalization of description length $L(\mathcal{C}, D)$ with the description length $L_r(\mathcal{C}, D)$, which uses prequential coding (see Sect. 3). Table 5 shows the results for the discovered clusterings using $L_r(\mathcal{C}, D)$ as the description length. We note that the numbers of discovered clusters are highly comparable overall. For the binary datasets we see that prequential coding is a bit more conservative in merging clusters, resulting in slightly higher numbers of clusters. For the categorical data there is virtually no difference, as there the shorter encoding of the 'left hand sides' of the code tables allows the efficient prequential encoding to merge more often than for binary data. Furthermore, the description lengths for both encodings are in the same ballpark.

**Table 5** Results of our algorithm using the description length $L_r(\mathcal{C}, D)$, which uses prequential coding

| | k | $L_r(\mathcal{C}, D)$ | $\frac{L_r(\mathcal{C},D)}{L_r(\mathcal{I},D)}$ (%) | $\frac{L_r(\mathcal{C},D)}{L_c(D)}$ (%) |
|---|---|---|---|---|
| Binary data | | | | |
| Independent | 48 | 894,855 | 99.9 | 89.5 |
| Markov | 11 | 880,567 | 88.1 | 88.1 |
| DAG | 8 | 774,276 | 82.5 | 77.4 |
| Accidents | 171 | 17,253,821 | 66.4 | 10.8 |
| BMS-Webview-1 | 97 | 1,045,076 | 87.3 | 3.5 |
| Chess | 15 | 60,892 | 42.7 | 25.4 |
| Connect | 15 | 1,632,662 | 45.4 | 18.7 |
| DNA Amplification | 76 | 77,598 | 41.2 | 4.3 |
| Mammals | 29 | 92,990 | 76.9 | 35.2 |
| MCADD | 23 | 2,050,896 | 72.1 | 32.4 |
| Mushroom | 18 | 248,494 | 56.1 | 25.7 |
| Pen Digits | 12 | 379,027 | 62.6 | 40.1 |
| Categorical data | | | | |
| Independent | 50 | 2,036,444 | 100.0 | 96.5 |
| Markov | 15 | 1,908,282 | 93.7 | 90.4 |
| Chess | 7 | 56,107 | 78.5 | 46.7 |
| Connect | 7 | 1,552,621 | 77.1 | 33.7 |
| MCADD | 7 | 1,756,006 | 89.3 | 81.0 |
| Mushroom | 6 | 188,540 | 70.7 | 48.6 |
| Pen Digits | 4 | 303,282 | 80.3 | 70.4 |

Shown are the number of discovered clusters $k$, and the absolute and relative description length with respect to the description length of the independence and canonical clusterings

It is important to emphasize, however, that we cannot simply compare the description lengths of two different clusterings under two different encodings. Nonetheless, the fact that the returned best clusterings for both encodings tend to be alike, and have a similar description length, is a good indication that our encoding is close to the theoretically clean refined MDL. Upon inspection, the discovered clusters are often comparable to the ones discovered using our $L(\mathcal{C}, D)$ description length, however, for many datasets the clusters discovered using $L_r(\mathcal{C}, D)$ tend to be of a slightly lower subjective quality. For instance, for the binary *Connect* data we do not obtain the seven column clusters that were obtained before. Consequently, since the results using our encoding are comparable or better than those using prequential coding, and additionally since our encoding is more intuitive, from this point on we will only be using our description length $L(\mathcal{C}, D)$.

### 7.4 Randomization

Next, we investigate whether our algorithm is generally capable of discovering structure beyond what can be straightforwardly explained by simpler statistics. As such,

for the binary datasets, we here consider the row and column margins, i.e., the number of ones per row and column. The idea is that if for some data the discovered attribute clustering simply follows from the margins of the data, then we would obtain the same result on any other dataset with the same margins. By considering random data with the same margins as the real datasets, we can so check whether our models are indeed able to model structure of the data at hand beyond what follows from its margins (assuming these datasets contain more structure than that). Note that for random data we generally expect our approach to return the independence clustering.

To obtain the random data samples needed for these experiments, we use swap randomization, which is the process of randomizing data to obscure the internal dependencies, while preserving the row and column margins of the data (Gionis et al. 2007). This is achieved by applying individual swap operations that maintain these margins. That is, one randomly finds two items $a$ and $b$, and two transactions, such that $a$ occurs in the first transaction but not in the second, and vice versa. Then, a swap operation simply swaps these items. This is a Markov Chain Monte Carlo process, which has to be repeated numerous times, as often as is required to break down the significant structure of the data, i.e., the mixing time of the chain; as suggested by Gionis et al. (2007), we use five times the number of ones in the data.

For each dataset we create 1,000 swap randomized datasets. We then calculate the average number of clusters our algorithm finds, and the average description length. Table 6 shows the results. We see that for all datasets the number of clusters is very close to the number of attributes, indicating that the structure that was present in the original datasets, was not simply a consequence of the row and column margins. Furthermore, the description lengths are much higher than those for the clusterings discovered in the original data. In fact, the third to last column of Table 6 shows that the average description length is almost exactly equal to the description length of the independence clustering. The last column of Table 6 shows the *empirical* $p$-value of our results, which is defined as the empirical probability of observing a description length at least as low as $L(\mathcal{C}, D)$, and is computed as

$$p = \frac{\left|\{D' \mid L(\mathcal{C}', D') \leq L(\mathcal{C}, D)\}\right| + 1}{|\{D'\}| + 1},$$

where $\{D'\}$ is the set of 1,000 swap randomized datasets. Figure 6 shows the description length distribution of the swap randomized versions of the *Connect* dataset. We see that the description length of the original dataset is significantly lower than that of any of the swap randomized datasets. The $p$-values show that this holds for all the binary datasets, except for *Independent*; the structure of this dataset can of course be completely described by the column margins only.

Our algorithm is greedy in nature, and therefore does not necessarily discover the optimal clustering, that is, the clustering $\mathcal{C}^*$ which globally minimizes $L(\mathcal{C}, D)$. To get an idea of how good the discovered clustering is, we compare it to random clusterings. We uniformly sampled 1,000 random $k$-attribute clusterings for each dataset (where $k$ is the size of the discovered clustering), and compute their description lengths. Table 7 shows for each dataset the absolute and relative average description length and standard deviation. For most datasets the relative description length is still less

**Table 6** Swap randomization experiments on the binary data, for 1,000 swap randomized datasets

| Binary data | $k$ | $L(\mathcal{C}', D')$ | $L(\mathcal{C}, D)$ | $\frac{L(\mathcal{C}', D')}{L(\mathcal{I}, D')}$ (%) | $\frac{L(\mathcal{C}', D')}{L_c(D')}$ (%) | Empirical $p$-value (%) |
|---|---|---|---|---|---|---|
| Independent | 49.2 | 895,078 | 895,078 | 99.9 | 89.5 | 51.5 |
| Markov | 48.9 | 999,157 | 884,943 | 99.9 | 99.9 | <0.1 |
| DAG | 48.9 | 970,481 | 797,588 | 99.9 | 97.0 | <0.1 |
| Accidents | 198.2 | 25,987,733 | 16,893,671 | 99.9 | 16.3 | <0.1 |
| BMS-Webview-1 | 221.1 | 1,195,928 | 1,078,905 | 99.5 | 4.0 | <0.1 |
| Chess | 58.8 | 142,748 | 58,457 | 99.9 | 59.5 | <0.1 |
| Connect | 80.9 | 3,592,591 | 1,559,773 | 99.9 | 41.2 | <0.1 |
| DNA Amplification | 194.1 | 190,422 | 82,209 | 99.5 | 10.6 | <0.1 |
| Mammals | 62.3 | 121,048 | 98,515 | 99.5 | 45.8 | <0.1 |
| MCADD | 163.7 | 2,844,348 | 1,816,628 | 99.9 | 45.0 | <0.1 |
| Mushroom | 77.5 | 443,042 | 169,425 | 99.9 | 45.8 | <0.1 |
| Pen Digits | 64.6 | 605,330 | 333,032 | 99.9 | 64.0 | <0.1 |

Shown are the average number of clusters $k$, the average absolute description length $L(\mathcal{C}', D')$, the description length $L(\mathcal{C}, D)$ of the original data, the average relative description length with respect to both the independence clustering description length $L(\mathcal{I}, D')$ and the canonical description length $L_c(D')$, and the empirical $p$-value for the original data

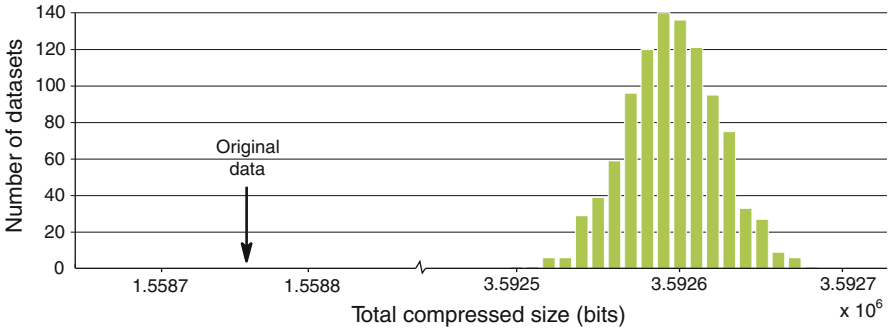


**Fig. 6** Distribution of the description lengths of 1,000 swap randomized version of the binary version of the *Connect* dataset. The description length of the original data is indicated by the *arrow*

than 100%, indicating that even random clusterings can capture *some* of the structure, especially for strongly structured datasets. However, we also see that for all datasets (except trivially for the categorical *Independent* data) the description length for random clusterings is much worse than that of the original discovered clustering (see Table 4). From the empirical $p$-values in the last column of Table 7, we can see that our algorithm significantly outperforms randomly generated clusterings. In fact, none of the random clusterings had a lower description length than the one discovered by our algorithm (again, except for *Independent*). Figure 7 depicts the description length distribution for the *BMS-Webview-1* dataset, of the 1,000 randomly generated random

**Table 7** Average description length over 1,000 randomly generated $k$-partitions, relative to the canonical description length of the datasets

| | $L(\mathcal{C}', D)$ | $L(\mathcal{C}, D)$ (%) | $\frac{L(\mathcal{C}', D)}{L_c(D)}$ (%) | Standard deviation (%) | Empirical $p$-value (%) |
|---|---|---|---|---|---|
| Binary data | | | | | |
| Independent | 930,878 | 895,078 | 93.1 | 0.3 | <0.1 |
| Markov | 994,857 | 884,943 | 99.5 | 0.7 | <0.1 |
| DAG | 976,662 | 797,588 | 93.2 | 1.1 | <0.1 |
| Accidents | 25,830,095 | 16,893,671 | 16.2 | 0.1 | <0.1 |
| BMS-Webview-1 | 1,200,869 | 1,078,905 | 4.1 | 0.0 | <0.1 |
| Chess | 135,785 | 58,457 | 56.6 | 1.9 | <0.1 |
| Connect | 3,274,147 | 1,559,773 | 37.6 | 0.9 | <0.1 |
| DNA Amplification | 199,770 | 82,209 | 11.1 | 0.1 | <0.1 |
| Mammals | 117,597 | 98,515 | 44.5 | 0.3 | <0.1 |
| MCADD | 3575766 | 1,816,628 | 56.6 | 1.9 | <0.1 |
| Mushroom | 341,035 | 169,425 | 35.3 | 0.7 | <0.1 |
| Pen Digits | 654,217 | 333,032 | 69.2 | 2.4 | <0.1 |
| Categorical data | | | | | |
| Independent | 2,037,016 | 2,037,016 | 96.5 | 0.0 | 100.0 |
| Markov | 2,240,147 | 1,932,611 | 115.9 | 9.2 | <0.1 |
| Chess | 70,424 | 57,353 | 59.6 | 1.0 | <0.1 |
| Connect | 1,916,691 | 1,554,827 | 41.6 | 0.6 | <0.1 |
| MCADD | 2,169,650 | 1,785,850 | 100.0 | 10.9 | <0.1 |
| Mushroom | 179,259 | 150,012 | 46.2 | 3.7 | <0.1 |
| Pen Digits | 399,095 | 309,788 | 92.7 | 7.8 | <0.1 |

Shown are the average absolute description length $L(\mathcal{C}', D)$, the description length $L(\mathcal{C}, D)$ of the original discovered clustering, the relative description length with standard deviation, and the empirical $p$-value of the original discovered clustering

attribute clusterings, together with the description length of the clustering found by our algorithm.

### 7.5 Beam search and simulated annealing

In this subsection we investigate whether we can improve our algorithm by employing different search strategies. We ran experiments using beam search for a range of beam widths. Table 8 shows the results for $b$ equal to 2, 5, 10, and 15. Note that using a beam width of 1 corresponds to our original algorithm. For most datasets and parameter settings, the discovered summaries are exactly the same as for a beam with of 1. In the other cases, some summaries with a lower description length were discovered, however, this relative decrease is barely noticeable (i.e., $10^{-3}$ or less). Meanwhile, it is clear that both runtime and memory consumption grow as the beam size is increased, namely by a factor $b$. Therefore, it does not seem to be favorable to add a beam to

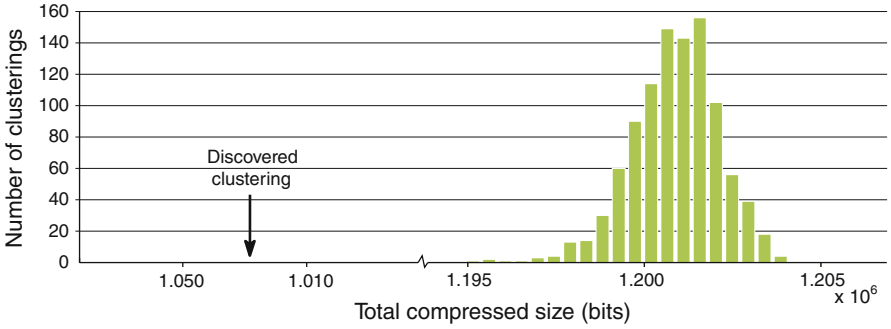


**Fig. 7** Distribution of the description lengths of 1,000 random $k$-clusterings, for the *BMS-Webview-1* dataset. The description length of the clustering discovered by our algorithm is indicated by the *arrow*

our algorithm, since neither the results nor the performance can notably be improved upon. The reason for this is that although beam search considers more search paths, the beam tends to exhibit very little variability and hence usually ends up with the same result.

Table 9 gives the results of similar experiments using simulated annealing. For each dataset, we execute 100 runs, where a single run consists of 1,000 iterations. The best clustering over those 100 runs is then reported. We repeat the experiments for varying settings of the $\lambda$ scaling parameter. For the synthetic datasets, we see that simulated annealing performs approximately the same as our algorithm, i.e., the compression ratios are comparable. For most of the other datasets, however, the results are noticeably worse than our algorithm. Only for *MCADD* does simulated annealing provide a marginally better result. The influence of the $\lambda$ parameter seems to be that the compression ratios decrease slightly for larger $\lambda$, although not much. This is because the $\lambda$ parameter controls the decrease of the probability threshold to move away from a local optimum. Due to the inherently nondeterministic nature of the simulated annealing algorithm, we must perform many runs, but we do not know how many in advance. Furthermore, in practice the simulated annealing algorithm has many more parameters than we used in our experiments, e.g., the cooling schedule, and tuning these parameters can prove to be difficult. Hence, simulated annealing does not seem to provide any improvement over our algorithm.

### 7.6 Frequency estimation

Finally, in this subsection we investigate how well our summaries can be used as surrogates of the data. We do this by using the code tables to estimate itemset frequencies, as described in Sect. 3.8. For each dataset we first mine the top 10,000 closed frequent itemsets.[3] Then, for each itemset in this collection, we estimate its frequency using the discovered attribute clustering, and compute both its absolute and relative error. For

[3] More precisely, we use the largest minimum support threshold such that the number of frequent closed itemsets is at least 10,000; therefore the total number of itemsets may be slightly larger.

**Table 8** Results for the beam search experiments for various beam widths

| | $\frac{L(\mathcal{C},D)}{L_c(D)}$ (%) | $b = 2$ (%) | $b = 5$ (%) | $b = 10$ (%) | $b = 15$ (%) |
|---|---|---|---|---|---|
| Binary data | | | | | |
| Independent | **89.5** | 89.5 | 89.5 | 89.5 | 89.5 |
| Markov | **88.5** | 88.5 | 88.5 | 88.5 | 88.5 |
| DAG | 79.8 | **79.7** | 79.7 | 79.7 | 79.7 |
| Accidents | **10.6** | 10.6 | 10.6 | 10.6 | 10.6 |
| BMS-Webview-1 | **3.6** | 3.6 | 3.6 | 3.6 | 3.6 |
| Chess | **24.4** | 24.4 | 24.4 | 24.4 | 24.4 |
| Connect | **17.9** | 17.9 | 17.9 | 17.9 | 17.9 |
| DNA Amplification | **4.6** | 4.6 | 4.6 | 4.6 | 4.6 |
| Mammals | **37.3** | 37.3 | 37.3 | 37.3 | 37.3 |
| MCADD | **28.7** | 28.7 | 28.7 | 28.7 | 28.7 |
| Mushroom | 17.5 | **17.4** | 17.4 | 17.4 | 17.4 |
| Pen Digits | **35.2** | 35.2 | 35.2 | 35.2 | 35.2 |
| Categorical data | | | | | |
| Independent | **96.5** | 96.5 | 96.5 | 96.5 | 96.5 |
| Markov | 91.6 | **91.4** | 91.4 | 91.4 | 91.4 |
| Chess | **47.7** | 47.7 | 47.7 | 47.7 | 47.7 |
| Connect | **33.8** | 33.8 | 33.8 | 33.8 | 33.8 |
| MCADD | 82.3 | 82.3 | **82.0** | 82.0 | 82.0 |
| Mushroom | 38.6 | 38.6 | **38.3** | 38.3 | 38.3 |
| Pen Digits | **71.9** | 71.9 | 71.9 | 71.9 | 71.9 |

The table gives the description length $L(\mathcal{C}, D)$ of the best discovered clustering, relative to the canonical description length $L_c(D)$. The first column repeats the results from Table 4, and is equivalent to the case $b = 1$. Model that compresses best depicted in bold at the smallest beam width for that score

comparison, we generate 1,000 random $k$-clusterings, estimate the itemset frequencies for each one, and average their mean absolute and relative error. The results are shown in Table 10.

Although frequency estimation is not the main goal of our approach, the results we obtain are very good. For most datasets, the average absolute error is less than 1%. Furthermore, the average relative error is usually also just a few percentage points. For the datasets where the relative error is larger, we see that the cause for this lies with the fact that the itemsets in those datasets have a very low average frequency. Compared to the random $k$-clusterings, we see that our algorithm always performs better on average, and this difference is significant, as can be seen from the empirical $p$-values in the last column. Further, as the code tables corresponding to a $k$-clustering—with $k < n$—inherently contain more information on how the attributes interact than for the independence clustering—which is an $n$-clustering, and hence by definition contains no correlations between attributes—our algorithm also performs better than the independence model in estimating count queries.

**Table 9** Results for the simulated annealing experiments for various settings of the λ parameter

| | $\frac{L(C,D)}{L_C(D)}$ (%) | $\lambda = 10^0$ (%) | $\lambda = 10^1$ (%) | $\lambda = 10^2$ (%) | $\lambda = 10^3$ (%) |
|---|---|---|---|---|---|
| Binary data | | | | | |
| Independent | **89.5** | 89.5 | 89.5 | 89.5 | 89.5 |
| Markov | **88.5** | 88.6 | 88.6 | 88.6 | 88.5 |
| DAG | **79.8** | 80.3 | 80.5 | 80.0 | 80.3 |
| Accidents | **10.6** | 15.8 | 15.8 | 15.6 | 15.7 |
| BMS-Webview-1 | **3.6** | 4.0 | 4.0 | 4.0 | 4.0 |
| Chess | **24.4** | 30.9 | 28.8 | 30.9 | 28.1 |
| Connect | **17.9** | 28.2 | 26.3 | 28.0 | 26.9 |
| DNA Amplification | **4.6** | 10.0 | 10.0 | 10.1 | 10.0 |
| Mammals | **37.3** | 40.4 | 40.4 | 40.6 | 40.2 |
| MCADD | **28.7** | 43.2 | 43.1 | 43.3 | 43.1 |
| Mushroom | **17.5** | 25.8 | 24.0 | 24.6 | 24.7 |
| Pen Digits | **35.2** | 47.1 | 48.2 | 48.7 | 48.1 |
| Categorical data | | | | | |
| Independent | **96.5** | 96.5 | 96.5 | 96.5 | 96.5 |
| Markov | **91.6** | 92.4 | 92.0 | 92.2 | 91.8 |
| Chess | **47.7** | 48.0 | 48.0 | 48.1 | 48.0 |
| Connect | **33.8** | 34.2 | 34.3 | 34.2 | 34.2 |
| MCADD | 82.3 | **82.1** | 82.1 | 82.1 | 82.1 |
| Mushroom | **38.6** | 39.7 | 39.5 | 39.7 | 39.7 |
| Pen Digits | **71.9** | 73.8 | 72.3 | 72.5 | 72.1 |

The table gives the description length $L(C, D)$ of the best discovered clustering, relative to the canonical description length $L_C(D)$. The first column repeats the results from Table 4. Best compressing model depicted in bold

Figure 8 shows the average estimation error in function of the frequency of the top-10,000 closed itemsets for the *Mushroom* dataset, in bins with a width of 5%. We see that the error tends to grow as the frequency decreases; the reason for this lies in the fact that low-frequency itemsets tend to be larger (as also depicted in the figure), and hence we have to combine information from more code tables, which makes the estimate less precise since in doing so we make more independence assumptions.

In Fig. 9 we plot the cumulative probability of the absolute estimation error for *Connect* and *Mushroom*, respectively. For every $\epsilon \in [0, 1]$ we determine the probability $\delta$ that the absolute estimation error $| fr(X) - \hat{fr}(X) |$ is greater than $\epsilon$. For both datasets we see that our summaries outperform the random $k$-clusterings, which in turn only marginally improve upon the independence model. For instance, for *Mushroom* we see that probability of an absolute estimation error larger than 5% is about 40% for the $k$-random model, whereas for our model this is only 1%.

In Table 11 the speed of querying code tables is demonstrated. For each dataset, we generate 100,000 itemsets uniformly, with sizes uniformly picked between 2 and the size of the largest transaction. First, we measure the query speed of a straightfor-

**Table 10** Frequency estimation of the top 10,000 closed frequent itemsets

| | $\overline{fr}$ (%) | Attribute clustering (%) | | Random $k$-partition (%) | | Empirical $p$-value (%) |
|---|---|---|---|---|---|---|
| | | $\lvert fr - \hat{fr} \rvert$ | $\frac{\lvert fr - \hat{fr} \rvert}{fr}$ | $\lvert fr - \hat{fr} \rvert$ | $\frac{\lvert fr - \hat{fr} \rvert}{fr}$ | |
| Binary data | | | | | | |
| Independent | 29.0 | 0.1 | 0.5 | 1.3 | 4.5 | <0.1 |
| Markov | 15.7 | 0.3 | 2.0 | 1.3 | 7.9 | <0.1 |
| DAG | 25.5 | 0.6 | 2.4 | 2.1 | 8.7 | <0.1 |
| Accidents | 55.8 | 1.4 | 2.7 | 2.9 | 5.3 | <0.1 |
| BMS-Webview-1 | 0.1 | 0.1 | 83.8 | 0.1 | 91.2 | <0.1 |
| Chess | 81.2 | 1.0 | 1.2 | 1.4 | 1.7 | 0.2 |
| Connect | 88.8 | 0.4 | 0.4 | 2.2 | 2.5 | <0.1 |
| DNA Amplification | 0.7 | 0.1 | 53.3 | 4.2 | 86.6 | <0.1 |
| Mammals | 43.0 | 13.3 | 31.7 | 19.8 | 46.8 | <0.1 |
| MCADD | 2.6 | 0.2 | 8.7 | 0.3 | 13.1 | <0.1 |
| Mushroom | 12.5 | 1.3 | 13.6 | 5.1 | 44.9 | <0.1 |
| Pen Digits | 6.1 | 2.9 | 52.1 | 3.3 | 58.5 | <0.1 |
| Categorical data | | | | | | |
| Independent | 10.2 | 0.1 | 1.3 | 0.1 | 1.3 | 100 |
| Markov | 11.7 | 0.3 | 2.7 | 0.6 | 5.1 | <0.1 |
| Chess | 81.2 | 0.5 | 0.6 | 1.3 | 1.6 | <0.1 |
| Connect | 88.8 | 0.4 | 0.4 | 2.2 | 2.5 | <0.1 |
| MCADD | 2.6 | 0.2 | 8.5 | 0.3 | 13.1 | <0.1 |
| Mushroom | 12.5 | 2.8 | 2.5 | 38.8 | 380.7 | <0.1 |
| Pen Digits | 6.1 | 3.0 | 53.8 | 3.3 | 58.9 | 1.3 |

Depicted are the average frequency $\overline{fr}$ of the itemsets in the original data, the average absolute and relative errors of the frequency estimates using our model, the average absolute and relative errors for 1,000 random $k$-partitions, and the empirical $p$-values of the result of our algorithm for the relative estimation error

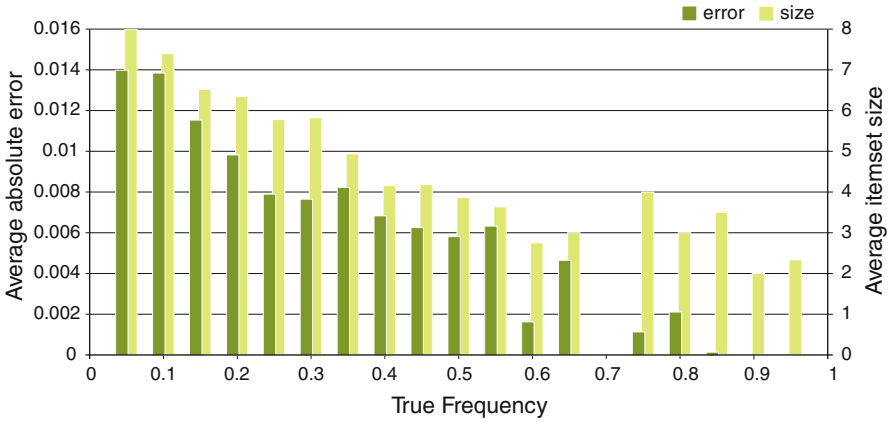


**Fig. 8** The average estimation error per frequency, and the average itemset size per frequency for the *Mushroom* dataset

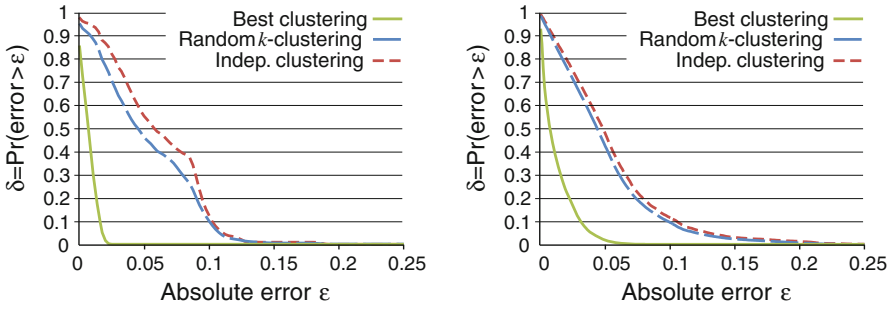


**Fig. 9** Probability of a frequency estimation error larger than $\epsilon$ for *Connect* (*left*) and *Mushroom* (*right*) on the top 10,000 closed frequent itemsets

**Table 11** The speed of direct querying versus frequency estimation, averaged over 100,000 randomly sampled itemsets, and the point at which the aggregate model construction and approximate querying time overtakes exact querying

| | Query | Summary | | Takeover |
|---|---|---|---|---|
| | | Construction | Estimation | |
| Binary data | | | | |
| Independent | 2.37 ms | 1 s | 0.02 ms | 740 |
| Markov | 2.27 ms | 4 s | 0.04 ms | 1,400 |
| DAG | 2.31 ms | 5 s | 0.11 ms | 2,450 |
| Accidents | 45.18 ms | 120 min | 2.07 ms | *166,968* |
| BMS-Webview-1 | 1.91 ms | 13 min | 0.29 ms | *596,913* |
| Chess | 0.32 ms | 1 s | 0.03 ms | 1,790 |
| Connect | 7.52 ms | 88 s | 0.06 ms | 15,220 |
| DNA Amplification | 0.24 ms | 33 s | 0.16 ms | *412,500* |
| Mammals | 0.20 ms | 2 s | 0.05 ms | 5,390 |
| MCADD | 2.49 ms | 146 s | 0.08 ms | 60,380 |
| Mushroom | 0.50 ms | 13 s | 0.05 ms | 28,140 |
| Pen Digits | 0.58 ms | 10 s | 0.10 ms | 31,660 |
| Categorical data | | | | |
| Independent | 0.53 ms | 4 s | 0.05 ms | 6,170 |
| Markov | 0.53 ms | 12 s | 0.11 ms | 28,510 |
| Chess | 0.09 ms | 1 s | 0.04 ms | 5,380 |
| Connect | 2.52 ms | 11 s | 0.07 ms | 5,810 |
| MCADD | 0.59 ms | 6 s | 0.06 ms | 9,030 |
| Mushroom | 0.17 ms | 1 s | 0.05 ms | 2,850 |
| Pen Digits | 0.21 ms | 1 s | 0.07 ms | 3,270 |

Values in italics are extrapolated

ward counting algorithm, which iterates over all transactions, and increases a counter whenever the queried itemset is contained in a transaction. Second, we measure the speed of querying by using code tables rather than the data itself. Both methods were

implemented in C++, compiled with the same compilation flags, and executed on the same machine, and are included in the provided implementation. The first and second columns of Table 11 show the average query time in milliseconds per itemset. For all datasets, using approximate querying is considerably faster, often an order of magnitude or more. Constructing the code tables, however, also takes time, and hence there is a point up to which querying the data is still cheaper. The rightmost column shows the number of queries such that the time taken to construct the model plus the time to perform the queries, is equal to querying the data (measured in a resolution of ten itemsets). For most datasets, the takeover point lies at a few thousand queries.

Lastly, we compare the estimation of our summaries to the profile-based summarization approach by Yan et al. (2005). A set of (overlapping) profiles summarizes a given set of patterns—rather than the data itself. A profile can be seen as a conditional independence distribution on a subset of the items, which can subsequently be queried. Although summarization with profiles is different from our clustering approach, we can compare the quality of the frequency estimates. We mimic the experiments by Yan et al. (2005) on the *Mushroom* and *BMS-Webview-1* datasets, by comparing the average relative error, also called *restoration error* in the paper. The collection of itemsets contains all frequent closed itemsets for a *minsup* threshold of 25 and 0.1% respectively. For *Mushroom* we attain a restoration error of 2.31%, which is lower than the results reported by Yan et al. (2005) for any number of profiles. For *BMS-Webview-1* our restoration error is 70.4%, which is on par with the results recorded by Yan et al. when using about 100 profiles. Their results improve when increasing the number of profiles. However, the best scores require over a thousand profiles, arguably not a feasible number of profiles to inspect by hand.

## 8 Discussion

The experiments show that our method discovers summaries of high quality. Their description lengths are noticeably low compared to both the canonical description lengths, as well as the independence encoding, which supports that they model important structure of the data. Moreover, inspection of the returned clusterings shows that on synthetic data the discovered attribute clusterings are in accordance with their respective generative processes, while on real data correlated attributes are correctly grouped, and the discovered clusterings are intuitive and logical. Through inspection of the code table of a cluster, we showed the user is given clear insight in the distribution of the data over the grouped attributes, and can easily identify the prevalent attribute-value combinations.

The comparison between the two encodings shows our basic two-part MDL encoding provides performance similar to or even slightly better than the theoretically more refined encoding, which tends to be more conservative. Additionally, the basic encoding has the advantage of being more interpretable. While our algorithm is greedy, and therefore does not necessarily produce an optimal result, experiments showed that the discovered summaries obtain significantly lower description lengths than for random clusterings. The swap randomization experiments validate that our approach identifies structure that can generally not be explained through simple statistics, i.e., the row and

column margins of the data. As such, we meet our goal that our summaries provide basic insight in the data that goes beyond first order statistics.

The experiments on alternative search strategies show that beam search and simulated annealing do not provide any noticeable improvement over our algorithm, which indicates that our greedy search indeed exploits the structure of the search space well. Finally, we demonstrated that besides the fact that attribute clusterings are a simple and intuitive way to gain insight into a dataset, they can also be used as a surrogate for the data that can be queried fast and reliably; itemset frequencies of the top 10,000 closed frequent itemsets are approximated with very high precision.

As such, we have shown that the summaries we discover provide useful insight into the data beyond simple first order statistics, and can hence be used as a quick first inspection of the basic structure of the data, for instance in order to decide whether and how the data should be analyzed in detail. Moreover, as they are simply groupings of attributes, and counts of the value combinations thereof, they can be straightforwardly included explicitly as background knowledge in other mining algorithms, in order to prevent discovery of results that can be explained by what we already know (Hanhijärvi et al. 2009; Mampaey et al. 2011).

While much of the focus in data mining research is on detailed data analysis, we point out the importance of lightweight methods providing basic insight in data, to the end of making an informed decision on whether and how to mine the data at hand. Although such methods clearly should be fast, it is important to stress that the ease of interpreting their results is key.

Besides basic insight, other possible uses for our summaries include feature selection and feature construction. While not the aim of this work, and hence we do not go into detail, one could consider only clustering attributes with the target label, in order to identify by MDL the set of features that together best describe the target. Alternatively, one could consider each cluster as a new feature, choosing either all, or only the most prevalent, value combinations as its values. A related interesting future use for our summaries would be fast approximate frequent itemset mining, as the possible combinations of attribute-values can be effectively enumerated by a-priori.

Although the experiments show that high-quality summaries are discovered, by the greedy nature of our approach we have no guarantees on how well we approximate the optimum. While this is an often-encountered problem in MDL, it may be worthwhile to investigate whether a connection can be made to well-studied optimization problems, such as SETCOVER, for which optimality bounds are known.

By using MDL to determine the optimal summary, our method is parameter-free: the amount and structure in the data determines what model is chosen as optimal. In general, MDL can be regarded as data hungry. That is, the more data is available, the better more complex correlations can be detected. In general, our method is best applied to data of at least 100s of rows.

Even though the experiments show that our algorithm is fast in practice, we see room for improvements. For instance, the algorithm can be trivially and massively parallelized, as well as optimized by using *tid*-lists. The main bottleneck of our approach is the first step of the clustering, where all pair-wise distances have to be computed; it would be worthwhile to develop bounds or heuristics to postpone calculation of distances between attributes that will not be considered for joining. However, we especially

regard the development of fast *approximate* summarization techniques for databases with many transaction and/or attributes as an important topic for future research, in particular as many data mining techniques cannot consider such datasets directly, but could be made to consider the summary surrogate.

In this work we only consider categoric data, for which it is difficult to construct a sensible distance measure. As such, it would be interesting to investigate whether our approach can be extended toward ordinal data, which would require non-trivial extension of the encoding. Another, related, as well as important, open problem is the generation of summaries for heterogeneous databases, e.g., consisting of both numeric and categorical attributes.

## 9 Conclusions

In this paper we introduced a method for getting a good first impression of a dataset with categorical attributes. Our parameter-free method builds summaries by clustering attributes that strongly correlate, and uses the MDL principle to identify the best clustering—without requiring a distance measure between attributes. The result offers an overview of which attributes interact most strongly, and in what value instantiations they typically occur. Furthermore, since they form probabilistic models of the data, these summaries are good surrogates for the data that can be queried efficiently and accurately.

Experiments show that our method provides high-quality results that correctly identify groups of correlated attributes, and can be used to obtain close approximations of itemset frequencies.

**Acknowledgments** The authors thank i-ICT of Antwerp University Hospital (UZA) for providing the MCADD data and expertise. Furthermore, the authors wish to thank the anonymous reviewers for their detailed and highly constructive comments. Michael Mampaey is supported by a Ph.D. grant of the Agency for Innovation through Science and Technology in Flanders (IWT). Jilles Vreeken is supported by a Post-Doctoral Fellowship of the Research Foundation—Flanders (FWO).

## References

Au W, Chan K, Wong A, Wang Y (2005) Attribute clustering for grouping, selection, and classification of gene expression data. IEEE/ACM Trans Comput Biol Bioinform 2(2):83–101

Baumgartner C, Böhm C, Baumgartner D (2005) Modelling of classification rules on metabolic patterns including machine learning and expert knowledge. Biomed Inform 38(2):89–98

Bringmann B, Zimmermann A (2007) The chosen few: on identifying valuable patterns. In: Proceedings of the IEEE international conference on data mining (ICDM'07), IEEE, pp 63–72

Calders T, Goethals B (2007) Non-derivable itemset mining. Data Min Knowl Discov 14(1):171–206

Chakrabarti D, Papadimitriou S, Modha DS, Faloutsos C (2004) Fully automatic cross-associations. In: Proceedings of the ACM SIGKDD international conference on knowledge discovery and data mining (KDD'04), pp 79–88

Chandola V, Kumar V (2005) Summarization—compressing data into an informative representation. In: Proceedings of the IEEE international conference on data mining (ICDM'05), IEEE, pp 98–105

Coenen F (2003) The LUCS-KDD discretised/normalised ARM and CARM data library. http://www.csc.liv.ac.uk/~frans/KDD/Software/LUCS-KDD-DN/DataSets/dataSets.html. Accessed October 2010

Cover TM, Thomas JA (2006) Elements of information theory, 2nd edn. Wiley, New York

Das G, Mannila H, Ronkainen P (1997) Similarity of attributes by external probes. In: Proceedings of the ACM SIGKDD international conference on knowledge discovery and data mining (KDD'97), pp 23–29

De Bie T (2011) Maximum entropy models and subjective interestingness: an application to tiles in binary databases. Data Min Knowl Discov 23(3):407–446

Dhillon I, Mallela S, Kumar R (2003) A divisive information theoretic feature clustering algorithm for text classification. J Mach Learn Res 3:1265–1287

Frank A, Asuncion A (2010) UCI machine learning repository. http://archive.ics.uci.edu/ml. Accessed March 2011

Garriga GC, Junttila E, Mannila H (2011) Banded structure in binary matrices. Knowl Inf Syst (KAIS) 28(1):197–226

Gionis A, Mannila H, Mielikäinen T, Tsaparas P (2007) Assessing data mining results via swap randomization. Trans Knowl Discov Data 1(3):1556–4681

Goethals B, Zaki MJ (2003) Frequent itemset mining implementations repository (FIMI). http://fimi.ua.ac.be. Accessed October 2010

Grünwald PD (2007) The minimum description length principle. MIT Press, Cambridge

Han J, Cheng H, Xin D, Yan X (2007) Frequent pattern mining: current status and future directions. Data Min Knowl Discov 15(1):55–86

Hanhijärvi S, Ojala M, Vuokko N, Puolamäki K, Tatti N, Mannila H (2009) Tell me something I don't know: randomization strategies for iterative data mining. In: Proceedings of the ACM SIGKDD international conference on knowledge discovery and data mining (KDD'09). ACM, New York, pp 379–388

Heikinheimo H, Hinkkanen E, Mannila H, Mielikäinen T, Seppänen JK (2007) Finding low-entropy sets and trees from binary data. In: Proceedings of the ACM SIGKDD international conference on knowledge discovery and data mining (KDD'07). ACM, New York, pp 350–359

Heikinheimo H, Vreeken J, Siebes A, Mannila H (2009) Low-entropy set selection. In: Proceedings of the SIAM international conference on data mining (SDM'09). SIAM, New York, pp 569–579

Kirkpatrick S (1984) Optimization by simulated annealing: quantitative studies. Stat Phys 34(5):975–986

Knobbe AJ, Ho EKY (2006) Maximally informative $k$-itemsets and their efficient discovery. In: Proceedings of the ACM SIGKDD international conference on knowledge discovery and data mining (KDD'06). ACM, New York, pp 237–244

Kontonasios KN, De Bie T (2010) An information-theoretic approach to finding noisy tiles in binary databases. In: Proceedings of the SIAM international conference on data mining (SDM'10). SIAM, New York, pp 153–164

Li M, Vitányi P (1993) An introduction to Kolmogorov complexity and its applications. Springer, New York

Mampaey M, Vreeken J (2010) Summarising data by clustering items. In: Proceedings of the European conference on machine learning and principles and practice of knowledge discovery in databases (ECML PKDD'10). Springer, New York, pp 321–336

Mampaey M, Tatti N, Vreeken J (2011) Tell me what I need to know: succinctly summarizing data with itemsets. In: Proceedings of the ACM SIGKDD international conference on knowledge discovery and data mining (KDD'11). ACM, New York, pp 573–581

Mitchell-Jones A, Amori G, Bogdanowicz W, Krystufek B, Reijnders PH, Spitzenberger F, Stubbe M, Thissen J, Vohralik V, Zima J (1999) The atlas of European mammals. Academic Press, London

Myllykangas S, Himberg J, Böhling T, Nagy B, Hollmén J, Knuutila S (2006) DNA copy number amplification profiling of human neoplasms. Oncogene 25(55):7324–7332

Pasquier N, Bastide Y, Taouil R, Lakhal L (1999) Discovering frequent closed itemsets for association rules. In: Proceedings of the ICDT international conference on database theory, pp 398–416

Pensa R, Robardet C, Boulicaut JF (2005) A bi-clustering framework for categorical data. In: Proceedings of the European conference on principles and practice of knowledge discovery in databases (PKDD'05). Springer, New York, pp 643–650

Rissanen J (1978) Modeling by shortest data description. Automatica 14(1):465–471

Rissanen J (2007) Information and complexity in statistical modeling. Springer, New York

Shannon CE (1948) A mathematical theory of communication. Bell Syst Tech J 27:379–423

Siebes A, Vreeken J, van Leeuwen M (2006) Item sets that compress. In: Proceedings of the SIAM international conference on data mining (SDM'06). SIAM, New York, pp 393–404

Vanden Bulcke T, Vanden Broucke P, Van Hoof V, Wouters K, Vanden Broucke S, Smits G, Smits E, Proesmans S, Van Genechten T, Eyskens F (2011) Data mining methods for classification of Medium-Chain

Acyl-CoA dehydrogenase deficiency (MCADD) using non-derivatized tandem MS neonatal screening data. J Biomed Inform 44(2):319–325

Vereshchagin N, Vitanyi P (2004) Kolmogorov's structure functions and model selection. IEEE Trans Inf Theory 50(12):3265–3290

Vreeken J, van Leeuwen M, Siebes A (2007) Preserving privacy through data generation. In: Proceedings of the IEEE international conference on data mining (ICDM'07), IEEE, pp 685–690

Vreeken J, van Leeuwen M, Siebes A (2011) KRIMP: mining itemsets that compress. Data Min Knowl Discov 23(1):169–214

Wallace C (2005) Statistical and inductive inference by minimum message length. Springer, New York

Wang J, Karypis G (2004) SUMMARY: efficiently summarizing transactions for clustering. In: Proceedings of the IEEE international conference on data mining (ICDM'04), IEEE, pp 241–248

Wang C, Parthasarathy S (2006) Summarizing itemset patterns using probabilistic models. In: Proceedings of the ACM SIGKDD international conference on knowledge discovery and data mining (KDD'06). ACM, New York, pp 730–735

Yan X, Cheng H, Han J, Xin D (2005) Summarizing itemset patterns: a profile-based approach. In: Proceedings of the ACM SIGKDD international conference on knowledge discovery and data mining (KDD'05). ACM, New York, pp 314–323