Low-Entropy Set Selection

Hannes Heikinheimo
HIIT, ICS Department
Helsinki University of Technology
hannes.heikinheimo@tkk.fi

Jilles Vreeken
Department of Computer Science
Universiteit Utrecht
jillesv@cs.uu.nl

Arno Siebes
Department of Computer Science
Universiteit Utrecht
arno@cs.uu.nl

Heikki Mannila
HIIT, University of Helsinki
Helsinki University of Technology
heikki.mannila@cs.helsinki.fi

Abstract

Most pattern discovery algorithms easily generate very large numbers of patterns, making the results impossible to understand and hard to use. Recently, the problem of instead selecting a small subset of informative patterns from a large collection of patterns has attracted a lot of interest. In this paper we present a succinct way of representing data on the basis of itemsets that identify strong interactions.

This new approach, LESS, provides a more powerful and more general technique to data description than existing approaches. Low-entropy sets consider the data symmetrically and as such identify strong interactions between attributes, not just between items that are present. Selection of these patterns is executed through the MDL-criterion. This results in only a handful of sets that together form a compact lossless description of the data.

By using entropy-based elements for the data description, we can successfully apply the maximum likelihood principle to locally cover the data optimally. Further, it allows for a fast, natural and well performing heuristic. Based on these approaches we present two algorithms that provide high-quality descriptions of the data in terms of strongly interacting variables.

Experiments on these methods show that high-quality results are mined: very small pattern sets are returned that are easily interpretable and understandable descriptions of the data, and can be straightforwardly visualized. Swap randomization experiments and high compression ratios show that they capture the structure of the data well.

Keywords: pattern subset selection, low-entropy sets, MDL, maximum likelihood principle, dense data

1 Introduction

One of the central research themes in data mining has been the discovery of frequently occurring patterns. Starting from frequent sets and association rules [1], one of the key goals has been completeness in discovery: the task is to find all patterns from a pattern class that satisfy certain conditions. This goal is, in a way, a very useful one: from the answer we know exactly every pattern that fulfils the condition.

The drawback is that the number of patterns returned is typically prohibitively large. Generally, there are lots of patterns satisfying the conditions, but many patterns convey roughly the same information about the data.

Recently, several authors have studied the pattern selection problem: given a large set of patterns, find a small subset of informative patterns. Examples of such work are [3, 13, 15, 19, 20]. These proposals all manage to reduce the pattern explosion significantly and achieve massive reductions in the number of patterns. However, whether these describe the data in full, or only partly, has a strong influence on the number of selected patterns: respectively up to hundreds, or only tens.

Lossy approaches, due to the small number of resulting patterns, allow for very easy interpretation. However, they cannot explain the data in full detail and may overlook important and interesting interactions. Lossless approaches, on the other hand, typically result into slightly more patterns. While this improved level of detail allows for thorough data analysis, interpreting or inspecting these groups of patterns by hand can be more difficult.

In this paper we provide a lossless method for succinct description of datasets using low-entropy itemsets. Our approach obtains very small collections of informative patterns, typically in the order of tens of patterns, that are both readable and provide intuitive descriptions of the data. The method is inspired by two recent approaches: low-entropy sets [10] and the MDL-based method KRIMP [19].

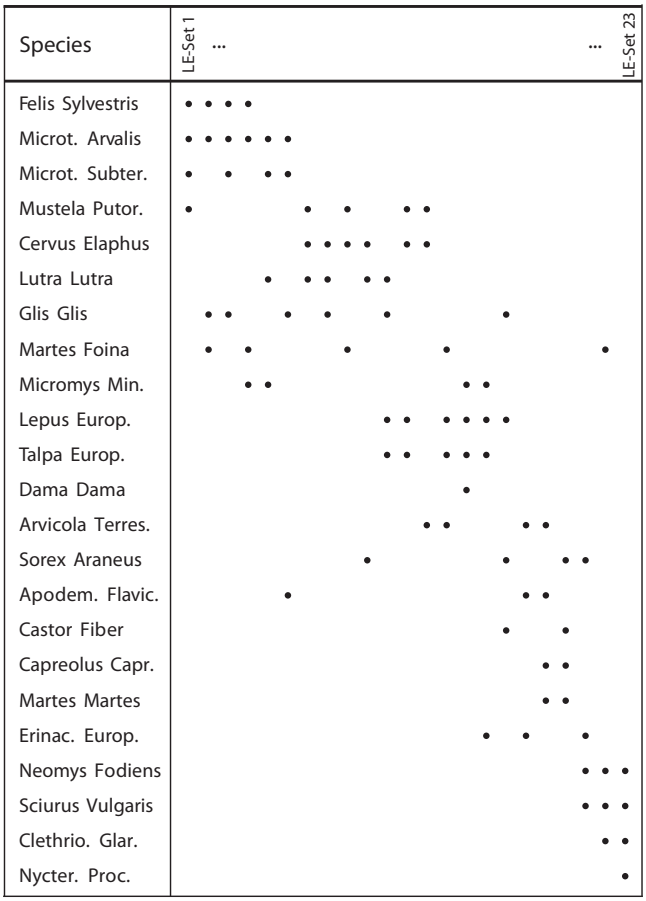

Figure 1: Visualization of 23 sets (columns) that our method selected to describe the occurrence interactions of 23 mammal species.

Informally, a low-entropy set is a set X of variables such that the distribution of the data on these variables is highly skewed, i.e., has low entropy. For example, consider a set $X = \{A,B,C,D,E\}$ of binary variables. Assume further that the data has 1000 rows where the values of these variables are $(1,0,0,1,0)$, and 2000 for which the values are $(1,0,1,0,0)$, and that the frequencies of the remaining 30 value combinations (2^5 in total) are all negligibly small. Together these variables interact strongly, i.e. their values are strongly structured, and resultantly the entropy of the dataset on the variables in X is small: for a single row of the data we can code the values of the variables in X using only few bits on average (about 0.9, in this case).

Low-entropy sets can be viewed as a stark generalization of frequent itemsets, which just look for sets X such that there are sufficiently many rows that have a 1 in each column of X. Unlike frequent itemsets, low-entropy sets are symmetric with respect to 0 and 1. As above, they can locate subsets that have just a few different dominant values. Therefore, they are very applicable for analyzing dense data. However, as with frequent sets, the number of low-entropy patterns can grow prohibitively large: for higher levels of entropy, many more sets are found than is practical for analysis by hand.

The MDL-approach [19] to selecting pattern subsets is based on the idea that the best subset of patterns is the one that compresses the data best. It identifies the best collection of itemsets as the one that requires the fewest bits to describe all of the data. For frequent itemsets, a transaction can be described simply by telling which itemsets together (i.e. their union) form the transaction.

For low-entropy sets this is less straightforward; to (re)construct a data row, we have to identify both the low-entropy sets and their specific variable values. For example, if a data row t would happen to have the combination $(1,0,1,0,0)$ on the variables of X, to describe t we can say that low-entropy set X is to be used, with $(1,0,1,0,0)$ as its value combination; as this combination is so frequent, it can be encoded in only a few bits. However, opposed to frequent itemsets, low-entropy sets can be used to describe *any* transaction: there always is one instantiation that fits the row. We are therefore required to use a fine-grained selection to determine which low-entropy set will be used to encode what part of the data. Turning this to our advantage, we provide two methods based on the maximum likelihood principle to optimally cover the data locally.

Using these principled selection strategies, we employ the MDL criterion to encode the whole data succinctly using low-entropy sets. As such, the method requires only tens of patterns for a detailed lossless description of the full data. Consequently, the outcome can be interpreted very easily.

As an example, consider Figure 1. It visualizes the results of our method on a dataset concerning the geographical presence of mammal species. In this picture, we show the low-entropy sets (the columns) that our method selected to describe the interactions between the 23 species. These sets were selected by our method out of the 67677 low-entropy patterns of entropy ≤ 3.3 bits. Indeed, it is a compact set of species interactions that together well describe the main essence of the data.

The reason why the data can be characterized by such a small number of patterns is the fact that one low-entropy set may capture multiple interactions at once for the same attribute set. Consider for instance the left–most column of Figure 1, showing an interaction pattern between two Vole species (*M. Arvalis* and *M. Subterraneus*) together with the predators Wild cat (*F. Sylvestris*) and European Polecat (*M. Putor.*). Table 1 takes a closer look at the usage counts of the individual variable combinations of the set as they are used to describe the data. As the table suggests, relevant interactions involve several occurrence combinations, as well as absence of the species. Hence, if the

F. Sylvestris	M. Arvalis	M. Suberraneus	M. Putor.	counts
0	0	0	0	44
0	0	0	1	10
0	1	0	0	18
0	1	0	1	199
0	1	1	1	248
1	0	0	0	5
1	0	0	1	7
total # usage				531

Table 1: Detailed view of how LESS uses the left-most low-entropy set of Figure 1 to encode the data. The set depicts major presence-interactions of four mammal species.

same interactions would have to be described using regular itemsets many separate sets would be required, instead of the single low-entropy set required here.

In this study we provide the methodological and algorithmic solutions necessary to use the MDL framework for low-entropy set patterns. Experiments show that the end result yields easily interpretable small collections of low-entropy sets. The quality of the mined pattern groups is first verified through compression. By swap randomization experiments we affirm that these sets grasp the significant structure in the data. Further, we provide evidence that these sets together describe multiple distributions by comparing them to the cluster centroids of the data. In summary, the results show that our method only requires as few patterns as lossy methods do to provide a high-quality lossless description of the data.

The roadmap of this paper is as follows. First we introduce some preliminaries on low-entropy sets and how MDL can be used to select the most interesting subset of low-entropy sets. In Section 3 we present the LESS algorithm for Low Entropy Set Selection, as well as a principled way of encoding the data locally optimally. Next, in Section 4 we empirically evaluate the proposed method. Related studies are discussed in Section 5. We round up with discussion and conclusions.

2 Problem Definition

In this section we introduce preliminaries and notations used in subsequent sections.

2.1 Low-Entropy Sets Let $\mathcal{I}$ be a set of 0–1 valued attributes. A *transaction* t over $\mathcal{I}$ is a binary vector of length $|\mathcal{I}|$. A dataset $\mathcal{D}$ is simply a bag of transactions, the number of which is denoted by $|\mathcal{D}|$. We denote *attribute sets*, i.e., subsets of $\mathcal{I}$, by X and Y. For singleton sets we omit the braces, e.g., we write A instead of $\{A\}$.

We use $\pi_A(t)$ to refer to the value of attribute A on row t (1 or 0). Given an attribute set X, we denote by $\pi_X(t)$ the projection of the transaction t onto X. In other words, $\pi_X(t)$ is a 0–1 vector of values $\pi_A(t)$ defined by the attributes $A \in X$.

Let Ω_X be the set $\{0,1\}^{|X|}$ of all 0-1 vectors of length $|X|$. We call the vectors $i \in \Omega_X$ the *instantiations* of the attribute set X. We say that the instantiation i fits transaction t iff $i = \pi_X(t)$. The probability $p_X(i)$ of an instantiation i is the relative support in $\mathcal{D}$ of the attributes X having the value of i. More formally,

$$p_X(i) = \frac{|\{t \in \mathcal{D} | i = \pi_X(t)\}|}{|\mathcal{D}|}.$$

Or, simply put, the fraction of transactions in $\mathcal{D}$ where i fits. For readability, we write $p(i)$ wherever X is clear from the context.

The entropy of an attribute set X in $\mathcal{D}$ is

$$H(X) = -\sum_{i \in \Omega_X} p(i) \log_2 p(i),$$

where $0 \log_2 0$ is assigned the value of 0 by convention.

Entropy is a measure of skewness in the occurrence distribution of instantiations of X. The lower the entropy, the more structured and more concentrated the instantiations in the database are. From a pattern mining point of view, attribute sets exhibitting structure in the form of low entropy can therefore be considered interesting.

DEFINITION 1. *Given an entropy threshold ϵ, an attribute set X is a low-entropy set (LE-set) in $\mathcal{D}$ if $H(X) \leq \epsilon$.*

It is straightforward to show that low-entropy sets have a monotonicity property. Say we combine attributes A and B into a set X. By definition, $H(X)$ is minimal iff X has no instantiations of lower probability than A or B separately. In other words, no value combination of A and B is more surprising than any of the value combinations of A or B separately. This is only the case if A and B are either exact copies or exact negatives, resulting in $H(X) = H(A) = H(B)$. Otherwise, if A and B disagree on one or more values we have $H(X) \geq H(A)$ and $H(X) \geq H(B)$. For the low-entropy mining task this monotonicity property allows to use e.g. a level-wise search in similar fashion to that of frequent items [1]. For a formal proof, and more details on mining low-entropy sets, see [10].

2.2 MDL for Low-Entropy Sets One can summarize the MDL approach to induction by the slogan: *the best model compresses the data best.* Slightly more formal, it can be described as follows: Given a set of models $\mathcal{M}$, the best model $M \in \mathcal{M}$ is the one that minimizes

$$L(M) + L(\mathcal{D}|M)$$

in which

- $L(M)$ is the length, in bits, of the description of M, and
- $L(\mathcal{D}|M)$ is the length, in bits, of the description of the data when encoded with M.

Note that this two-component approach is called *crude MDL* in [9]. The reason we use this version is that we are particularly interested in the set of LE-sets that yield the best compression. That is, we are especially interested in *how* the this best compression is reached.

Constructing a compression scheme that is based on LE-sets is not trivial. Unlike for itemsets [19], unambiguous decoding is impossible if only the sets are encoded: we also have to identify which individual instantiations are used. Here, we want to describe the data primarily in terms of low-entropy sets, and are less interested in their value instantiations. Therefore those value identifying codes should provide as little as possible bias to which LE-sets are chosen, while at the same time the complexity of the model should be weighed properly. This, we reach by encoding the sets and the instantiations separately. Most importantly, we make the code lengths for the instantiations independent from those of the LE-sets.

The basic idea is as follows: to describe a transaction t, we tell which LE-sets and which instantiations are used to obtain the values $\pi_A(t)$ for each attribute A. During compression a *code table* is induced, a two-column table containing a list of LE-sets and the codes used to identify them. The codes come from a prefix-code to allow for unambiguous decoding. The more often a set is used to encode the transactions in the database, the shorter its associated code.

Example. Given a transaction t we code it by giving a sequence of LE-sets and instantiations for these. As a simple example, consider the attributes $\{ABCD\}$ and a transaction $t = \{A, D\}$ (i.e., the vector $(1, 0, 0, 1)$). This transaction can be described by the LE-sets $\{A, B\}$ and $\{C, D\}$ with instantiations $(1, 0)$ and $(0, 1)$ respectively. Let the code associated with $\{A, B\}$ be c_1 and let c_2 be the code associated with $\{C, D\}$. The naïve way to store this would be c_1c_2 for the coded transaction and $((1, 0), (0, 1))$ as the indication of the values. This last part is simply a representation of t as a binary vector. Although a possible encoding, as it completely ignores any structure it is hardly a way to compress.

So, we have to refine the encoding of the instantiations. Note that $\{(0, 0), (0, 1), (1, 0), (1, 1)\}$ is the set of instantiations of all two element LE-sets. Hence, one of these will be used whenever a two element LE-set is part of the cover of a transaction. Therefore we assign a (prefix) code to each of these instantiations; again, the more often an instantiation is used, the shorter its code. The codes for instantiations are called indicators. Continuing the example, let l_1 be the code associated with $(1, 0)$ and l_2 the one with $(0, 1)$. The transaction $\{A, D\}$ is then encoded by the pair of codes c_1c_2 and l_1l_2.

From the example follows that to compress transactions we require two code tables. The first, the LE-set code table denoted by CT_{LE}, is defined as a two-column table of which the first column contains LE-sets and the second contains their codes. Second, the indicator code table, denoted by CT_I, is analogously defined as a two-column table of which the first column contains indicators and the second column the associated codes. The encoding of a database $\mathcal{D}$ with this pair of code tables results in a pair of encodings. The first, $\mathcal{D}_{LE}$, contains the codes from CT_{LE}, the second, $\mathcal{D}_I$ contains the codes from CT_I.

As done in [19], we also require CT_{LE} to contain at least the singleton attribute sets, such that all possible transactions can be encoded using any valid code table. Similarly, if the largest LE-set in CT_{LE} contains n attributes, CT_I is defined to contain all possible indicators for one-element LE-sets up to those for n-element LE-sets. That is, CT_I will have $2^{n+1} - 1$ entries.

2.3 Coding the Transactions To determine the appropriate code for the elements of both CT_{LE} and CT_I we need to know how often an LE-set and its instantiations are used. That is, we have to define which elements of CT_{LE} are used to cover a transaction t and which instantiations of those elements are used.

Informally, a *cover function* provides a set of non-overlapping LE-sets such that they describe all attributes $\mathcal{I}$ of a transaction t of database $\mathcal{D}$. We formalize this as follows:

DEFINITION 2. *A* cover *function is a function that given a LE-set code table CT_{LE} and an indicator code table CT_I assigns to each transaction t a list of pairs*

$$(X, i) \;\; X \in CT_{LE}, i \in CT_I$$

such that the union of the instantiated LE-sets equals t. Slightly abusing notation, we write both $X \in cover(t)$ and $i \in cover(t)$ whenever $(X, i) \in cover(t)$.

Since the CT_{LE} elements in the result of a cover function are non-overlapping, *cover* only needs to return a list of CT_{LE} elements. The associated indicators can easily be reconstructed by considering this list and the transaction.

The number of times a CT_{LE} element X is used in the cover of a transaction $t \in \mathcal{D}$ is called its *frequency*. The frequency of an indicator $i \in CT_I$ is defined similarly:

$$\begin{aligned} freq(X) &= |\{t \in \mathcal{D} |\; X \in cover(t)\}| \\ freq(i) &= |\{t \in \mathcal{D} |\; i \in cover(t)\}| \end{aligned}$$

The probability that X or i is used in the cover of a randomly selected transaction t is thus

$$\begin{aligned} P(X|\mathcal{D}) &= \frac{freq(X)}{\sum_{Y \in CT_{LE}} freq(Y)}, \\ P(i|\mathcal{D}) &= \frac{freq(i)}{\sum_{j \in CT_I} freq(j)}. \end{aligned}$$

To compress the database optimally, we use a Shannon code [9] for both code tables. That means that the length (in bits) of the codes for X and i is

$$\begin{aligned} L(code(X)) &= -\log(P(X|\mathcal{D})), \\ L(code(i)) &= -\log(P(i|\mathcal{D})). \end{aligned}$$

Note that we are only interested in the lengths of these codes, not the actual codes themselves. Then, we can calculate the size of the encoded databases $\mathcal{D}_{LE}$ and $\mathcal{D}_I$, encoded respectively by CT_{LE} and CT_I, as

$$\begin{aligned} L(\mathcal{D}_{LE}) &= \sum_{X \in CT_{LE}} -freq(X)\log(P(X|\mathcal{D})), \\ L(\mathcal{D}_I) &= \sum_{i \in CT_I} -freq(i)\log(P(i|\mathcal{D})). \end{aligned}$$

The encoded size of the full database then is

$$L(\mathcal{D}) = L(\mathcal{D}_{LE}) + L(\mathcal{D}_I). \tag{2.1}$$

For the two code tables, we already know the size of the codes, viz., $L(code(X))$ and $L(code(i))$ as defined above. To compute, respectively, the sizes of the LE-sets and the indicators these codes stand for, we have to define how we encode them.

For CT_{LE}, we encode the LE-sets by what we define as the *standard* code table ST, which is the simplest valid code table: the code table that only contains the singleton attribute sets. Hence, the size of CT_{LE} is (ignoring elements $X \in CT_{LE}$ with $freq(X) = 0$)

$$L(CT_{LE}) = \sum_{X \in CT_{LE}} L(code(X)) + L(code_{ST}(X)).$$

For CT_I, we simply use the bit-representation of the instantiations. That is, the instantiation $(0, 1)$ is represented by 01. We denote the bit-representation of $i \in CT_I$ by $bit(i)$. Note that if the largest LE-set in CT_{LE} has n items, then

$$\sum_{i \in CT_I} bit(i) = \sum_{j=1} nj2^j = 2 + (n-1)2^{n+1}.$$

Hence, the size of CT_I is

$$L(CT_I) = \sum_{i \in CT_I} L(code(i)) + bit(i).$$

Then, we have as the total size for the code tables,

$$L(CT) = L(CT_{LE}) + L(CT_I).$$

2.4 The Formal Problem Statement Now that all the details of MDL for LE-sets have been defined, we can formally state our problem.

Let $\mathcal{D}$ be a transaction database, and cover *a cover function. Find the code tables CT_{LE} and CT_I minimizing the* total encoded size

$$L(\mathcal{D}, CT) = L(\mathcal{D}) + L(CT).$$

So, the actual problem is now to find the best code table. Note that given CT_{LE} and a *cover* function determining CT_I is trivial.

Still, the search space we have to consider for this problem is huge. First, it consists of all possible code tables CT_{LE}: all possible subsets of $\mathcal{P}(\mathcal{I})$ that contain at least the singleton sets $\mathcal{I}$. So, there are

$$\sum_{k=0}^{2^{|\mathcal{I}|}-|\mathcal{I}|-1} \binom{2^{|\mathcal{I}|} - |\mathcal{I}| - 1}{k}$$

possible code tables. In order to determine which one minimizes the total encoded size, we have to consider these using every possible cover function. This translates to using every possible cover order per transaction. Since there are $n!$ possible orders for a set of length n, the total size of the search space is

$$\sum_{k=0}^{2^{|\mathcal{I}|}-|\mathcal{I}|-1} \left(\binom{2^{|\mathcal{I}|} - |\mathcal{I}| - 1}{k} \times (k + |\mathcal{I}|)! \times |\mathcal{D}| \right).$$

In short, it is prohibitively large. To make matters worse, there is no useable structure that allows us to prune this search space. Hence, we need to use heuristics.

3 Algorithms

Our approach for finding the best possible code table can be divided into two main important elements:

- *transaction encoding phase*, where a good compression for each transaction (cover) is found using the patterns in the code table.
- *the search strategy*, which is the way in which the search space of all possible pattern subsets is traversed to find a good code table.

The method follows the general framework of [19]. However, we apply very different technical solutions within the different parts of the approach. We will discuss transaction encoding in Subsection 3.1 and the search strategy in 3.2.

3.1 Transaction Encoding As discussed in Section 2.2, when encoding the database the task is to compress the data as well as possible using only the LE-sets in the code table. This is done by encoding each transaction with a set of patterns from the code table. However, as pointed out in that section, an LE-set always has an instantiation that can be used to describe a transaction. Therefore, there are often many different ways (depending on the order in which we use sets from the code table) that a transaction can be covered, and hence compressed.

Our strategy to select good covers for each transaction is to take advantage of the statistical nature of low-entropy patterns and use the maximum likelihood (ML) principle. The idea is to take the cover C that maximizes the conditional probability $p(t|C)$ (likelihood) of the transaction given all possible covers. We define the likelihood of transaction t as follows:

DEFINITION 3. *Let t be a transaction and C a cover of t.*

$$llh(t, C) = \sum_{X \in C} \log p(\pi_X(t)) \tag{3.2}$$

is the loglikelihood of t given C.

For each transaction t, the task is then to find the optimal cover

$$C^* = \arg\max_C llh(t, C)$$

from the set of all possible covers, such that llh is maximized.

Maximum likelihood is a widely used and well principled way of selecting between alternative models for the data. That is, in our case to choose between different covers of a transactions. Moreover, the ML-principle has an intuitive connection to the overall task of minimizing the total encoded length of the data.

In more detail, the connection is as follows. First, let us consider a set of LE-sets that form a cover C of some transaction t. In order for C to provide a good compression of both this transaction and the whole of the data, the associated codes should be as short as possible. Hence, both the LE-sets $X \in C$ and the instantiations $\pi_X(t)$ should be used as often as possible. Now, let's assume that for all the rest of the transactions in the data the instantiation $\pi_X(t)$ and LE-set $X \in C$ are used to cover every transaction where $\pi_X(t)$ fits, and that the instantiation $\pi_X(t)$ is not used anywhere else. That is, more formally for the case of the instantiation,

$$freq(\pi_X(t)) = |\mathcal{D}| \cdot p(\pi_X(t)),$$

where $p(\pi_X(t))$ is the frequency of the transactions for which $\pi_X(t)$ fits. From this it follows that the code length of $\pi_X(t)$ will be strictly proportional[1] to its negative loglikelihood. More formally,

$$L(code(\pi_X(t))) \propto -log\frac{freq(\pi_X(t))}{|\mathcal{D}|} = -\log p(\pi_X(t)).$$

Therefore, optimizing equation 3.2 also optimizes the entire coding length of transaction t. Assuming this for every transaction in the database, the encoding size of the entire database will be proportional to the negative loglikelihood of the data.

The above assumption will not strictly hold in every case. However, in general patterns with instantiations of high likelihood (high support) are likely to behave approximately like this and will consequently optimize the encoded length of the data well.

3.1.1 Finding the Best Cover It is quite clear that finding an optimal cover for a transaction is NP-complete [7] and hence exact solutions cannot be computed for large datasets. However, many covering problems are known to be well approximable using greedy heuristics.

In this subsection, we first study covering a transaction optimally according to the maximal likelihood principle, with an exhaustive search strategy using pruning. In the next subsection, we discuss a greedy heuristic that can be applied for larger datasets.

To compute the optimal cover for transaction t, we start with a code table that is ordered on the per attribute likelihood addition. The idea is that given a transaction t we assign to LE-set $X \in CT_{LE}$ a weight $w(t, X)$ that is equal to the per attribute addition in likelihood that X would give, if it were to be added to the cover. More formally

$$w(t, X) = \log\big(p(\pi_X(t))\big)/|X|. \tag{3.3}$$

Given this ordered code table, we need to enumerate all possible covers in a depth first manner. We start from LE-set X with the largest weight $w(t, X)$ and greedily continue to add, non-overlapping, sets to the cover in decreasing order; backtracking the search at each time when reaching a full cover.

By taking this order into account, we can cut the search space down considerably using the following proposition.

[1] Notice that in Subsection 2.3 the probability $P(i|\mathcal{D})$ is normalized with the cover frequencies of all of the elements in CT_I instead of the size of the data $|\mathcal{D}|$. Hence, the code length is proportionally but not exactly the same as the negative loglikelihood.

Algorithm 1 The OPTIMALCOVER Algorithm

OptimalCover$(\mathcal{I}, CT, t)$:
OrderOnLikelihoodPerAttribute(CT, t)
return Optimal($\mathcal{I}$,CT,$\emptyset$,$\emptyset$)

Optimal$(\mathcal{I}, CT, C, BestC)$:
if $|C| = |\mathcal{I}|$ **then**
  return C
end if
$\Delta = \emptyset$
$m = |\mathcal{I}| - |C|$
for $e \in CT$ **do**
  $\Delta = e \cup \Delta$
  if $e \cap C = \emptyset$ **then**
    if $m \cdot w(e) + llh(C) > llh(BestC)$ **then**
      $CandC$ = Optimal($CT \backslash \Delta$,$C \cup e$,$BestC$)
      $BestC$ =arg min$(llh(BestC), llh(CandC))$
    end if
  end if
end for
return $BestC$

PROPOSITION 3.1. *Consider covering transaction t with disjunct attribute sets in strictly decreasing order according to the weight function w. Now, when at the* ith *attribute set, already having covered attributes Y with cover C_Y, we know that the resulting cover will have an apriori coding length of at most*

$$llh(t, C) \leq m \cdot w(t, X_i) + llh(t, C_Y), \qquad (3.4)$$

where m is the number of previously uncovered attributes.

The proposition follows straightforwardly from the fact that, if X_i covers all the previously uncovered attributes without overlapping Y, the likelihood of the resulting covering will be exactly the right hand side of inequality 3.4. Otherwise, we'll have to include some other set, which by the ordering on w will provide an equal or smaller addition in likelihood per attribute. Hence, this will result in a smaller overall likelihood for the transaction. Thus, Proposition 3.1 defines an upper bound that we can compare to the best found solution so far; and thus to decide whether it makes sense to continue building the current cover or to start backtracking already.

Written in pseudo code, this optimal covering approach is depicted in Algorithm 1. However, as the optimal covering method considers a prohibitively large search space, it only makes sense to apply it to moderately sized databases of up to about 25 attributes. To allow for more practical application, we present a fast, heuristic alternative that follows very naturally from the minimum apriori encoding length/maximum likelihood principle.

Algorithm 2 The GREEDYCOVER Algorithm

GreedyCover$(\mathcal{I}, CT, t)$:
OrderOnCodelengthPerAttribute(CT, t)
$Cover = \emptyset$
for $e \in CT$ **do**
  if $e \cap Cover = \emptyset$ **then**
    $Cover = e \cup Cover$
    if $|Cover| = |I|$ **then**
      return $Cover$
    end if
  end if
end for

3.1.2 Approximating the Best Cover Recall that our goal is to cover using elements that provide as high a gain as possible in the overall likelihood. The initial order used by the optimal algorithm provides us a very nice approximation, as it orders the elements on the gain in likelihood per attribute. If we use this order in a greedy fashion (without overlap), the resulting cover is the same as the first full cover the optimal cover strategy considers.

We present, as Algorithm 2, the translation of this simple scheme into pseudo code. Note that for an actual implementation a lot of speed can be gained as one can easily cache the per-transaction orders. As code tables remain very small this is fully feasible.

3.2 Search Strategy To cut down a large part of the search space, we use the following simple greedy search strategy:

- Start with the code table consisting only of the singleton attribute sets
- Add the low-entropy sets one by one. If the resulting codes lead to a better compression, keep it. Otherwise, discard the set.

By its iterative nature, the success of this strategy largely depends on the order in which the patterns are considered.

3.2.1 Ordering the Candidate Sets Using the strategy above, the optimal compression can be approximated best by trying all possible orders. However, as the number of possible orders of a set of size n equals $n!$ this clearly is infeasible for but the smallest of pattern collections. So, our last step to reduce the search space of our problem heuristically is to introduce an order on the candidate set J. We order the candidates such that sets that have a good chance of being used – those with high likelihood addition over size – are at the top of the list. On the candidate set level, this translates into preferring sets that have a low entropy over size, or $H(X)/|X|$. For a set J of low-entropy sets, **CandidateOrder**(J) returns the version of J in this order.

Algorithm 3 The LESS Algorithm

LESS$(\mathcal{I}, J, \mathcal{D})$:
J = CandidateOrder(J)
CT = CreateStandardCodeTable$(\mathcal{I}, \max|j \in J|)$
$compressedSize = L(\mathcal{D}, CT)$
for candidate low-entropy set $c \in J$ **do**
  $CT = CT \cup c$
  if $L(\mathcal{D}, CT) < compressedSize$ **then**
    $compressedSize = L(\mathcal{D}, CT)$
  else
    $CT = CT \setminus c$
  end if
end for
return CT

3.3 The Low-Entropy Set Selection Algorithm Now the main ingredients for our low-entropy set based compression algorithm are in place, we can assemble these into the Low-Entropy Set Selection (LESS for short) algorithm. We present it in pseudo–code as Algorithm 3.

As input, it requires the attribute set $\mathcal{I}$, the candidate set of low-entropy sets J, and a database $\mathcal{D}$. Also, one of the above discussed cover strategies for transaction encoding has to be chosen. Our naive compression process starts with the simplest description of the data, using only singletons to encode the data, together with the fully initialized indicator code table. Then, iteratively (in **CandidateOrder**) low-entropy sets are added to the code table one by one. Each time, using this new code table the new total compressed size of the database is calculated. If this addition improves the attained compression, the set is kept, otherwise it is permanently discarded.

In the course of this iterative process it is very well possible that by adding a new element, the usage of other patterns in the code table suddenly strongly decreases; thereby increasing their code lengths and possibly hindering overall compression. We therefore introduce a pruning variant of our method. Once an element of J is accepted into the code table, we reconsider all other elements $e \in CT_{LE}$ iteratively by temporarily removing them and calculating the compressed size. By MDL-principle, we then go for the best compression, permanently removing those elements that no longer help the compression.

4 Experiments

In this section we experimentally evaluate our methods. We first investigate the differences between OPTIMALCOVER and GREEDYCOVER. Next, we evaluate whether our method models relevant structure of the data. Thirdly, we look at the size of the resulting pattern sets and compare these to two other existing methods. Last, we examine these pattern sets in detail.

Dataset	$\lvert\mathcal{I}\rvert$	$\lvert\mathcal{D}\rvert$	*density*	$L(\mathcal{D}, ST)$
adult	97	48842	15.3	34229566
course	83	2405	20.5	1422594
heart	50	303	28.0	134588
letter recog	102	20000	16.7	14954124
mammals	40	2183	47.0	552457
mammals_{20}	20	2183	53.1	248665
mushroom	119	8124	19.3	7898102
pen digits	86	10992	19.8	6757243

Table 2: Statistics of the datasets used in the experiments. Per dataset the number of attributes, the number of transactions, the density (percentage of 1's) and the number of bits required by LESS to compress the data using the singleton-only standard code table ST.

4.1 Datasets For the experimental validation of our methods we use a wide range of datasets. From the widely used UCI repository [5] we take some of the largest and most dense databases. Further, we use two databases for which we know low-entropy analysis is well suited: the *mammals* and Helsinki CS-*course* databases. The former consists of presence/absence records of European mammals[2] within geographical areas of 50x50 kilometers [16]. The *course* data describes courses taken by students at the Department of Computer Science of the University of Helsinki. As we want to focus on interesting variable interactions, we disregard attributes with extremely high (about 1.0) or very low (about 0.0) support.

The details for these datasets are depicted in Table 2. For each database we show the number of attributes, the number of rows and the density: the percentage of 'present' attributes. The next column indicates the total compressed size in bits by using the singletons – only standard code table ST.

Due to the high density most of these datasets, they are not well suited for analysis by frequent itemset mining: far too many co-occurences exist. For example, at 10% support already over 11 million frequent itemsets are discovered in the *mammals* dataset and up to 5.5 billion can be extracted from the *mushroom* data.

When mining for low-entropy sets, those itemsets of which the attributes are too weakly correlated (i.e. their entropy is above the threshold) are ignored. In order to compress the data optimally, we have to allow LESS to consider as many low-entropy sets as possible. We therefore set the entropy threshold ϵ as low as feasible with our current low-entropy set mining implementation.

[2]The full version of the mammal dataset is available for research purposes upon request from the Societas Europaea Mammalogica. http://www.european-mammals.org

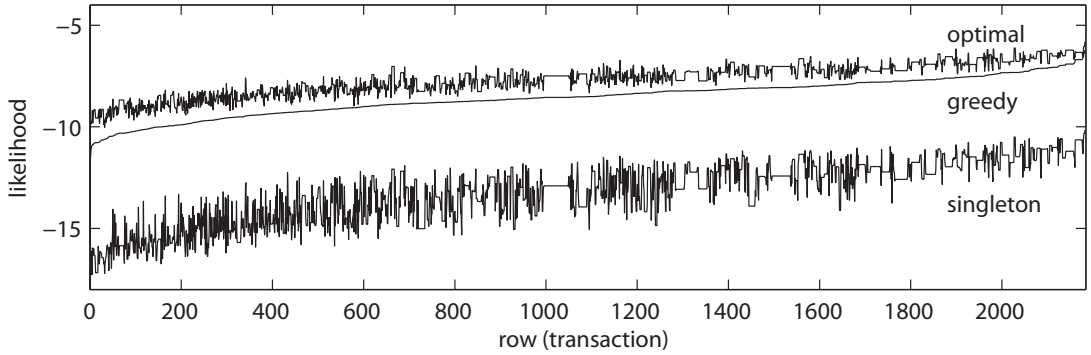

Figure 2: The likelihood scores for each transaction in the $mammals_{20}$ dataset using the OPTIMALCOVER Algorithm, the GREEDYCOVER Algorithm and singleton (all attributes independent) covering. The transactions have been sorted according to the likelihood score of GREEDYCOVER.

4.2 Optimal and Greedy Covering To compare the performance between the optimal and the greedy covering strategies (Algorithms 1 and 2), we use $mammals_{20}$, which constains the 20 most varying attributes of the full *mammals* dataset. On this data we mine low-entropy sets using a maximum entropy threshold of $3\frac{1}{3}$ bits, resulting in 2321 low-entropy sets.

For each transaction we compute a cover using both GREEDYCOVER and OPTIMALCOVER on all of the 2321 sets as the code table, as well as a baseline cover of only singleton sets. The results for each single transaction are presented in Figure 2. First of all, we see that using sets pays off both for the optimal and the greedy methods with an increase in log–likelihood for each transaction. Moreover, we see that the optimal transaction cover does indeed result in the highest log–likelihood scores. However, the much faster greedy approach finds covers that approximate the optimal score closely.

Next, we test these strategies with LESS using the LE-sets as candidates, but without pruning. In two hours, the optimal variant selected 25 sets to compress the data into 184572 bits. In one minute, the greedy approach finds a description of only 163908 bits, using 148 sets. By using a larger number of sets, it attains a higher likelihood over the data (-22091 and -19767, respectively). Analyzing the resulting code tables, it is evident that the optimal method is more picky and less promiscuous: if a set is allowed into the code table, it will stay in use and will not be fully traded in, opposed to what the greedy method does. However, the resulting code tables are very similar to those of the greedy algorithm when pruning is *enabled*. The greedy approach seems to concatenate the 'optimal' sets together into 12 sets to achieve a likelihood of -22330, while using only 145940 bits. Overall, the greedy cover algorithm allows MDL to condense the data better. When considered together, this tells us that GREEDYCOVER can be used as a fast and high quality alternative for OPTIMALCOVER. For the remainder of this section, we'll therefore use GREEDYCOVER.

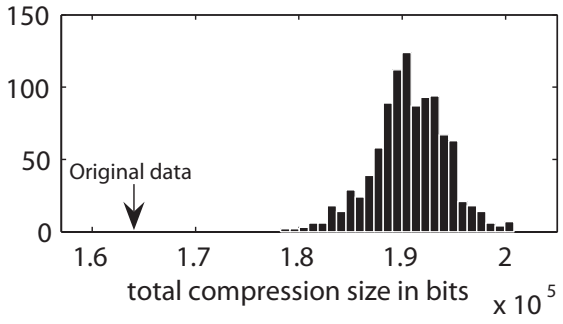

Figure 3: Distribution of the compressed sizes of the swap randomized $mammals_{20}$ datasets. The compressed size of the original dataset as indicated by the arrow, and is 1.64×10^5 bits.

4.3 Modeling Relevant Structure To evaluate whether LESS models the relevant structure in the data, we compare the compression scores of the actual data to those obtained from 1000 swap randomized [8] versions of that data. This process preserves the row and column margins of the given data set, but obscures the internal dependencies of the data. The idea is that if the true structure in the data is captured, there should be significant differences between the models found on the original and randomized datasets.

For this large number of experiments we used the $mammals_{20}$ dataset. We applied as many swaps as there are 1's in the data. Figure 3 shows the histogram of the compressed sizes of these 1000 databases.

The picture shows that the original data can be compressed significantly better than that of the randomized datasets (p-value of 0). Also, we noted that the loglikelihood

	Candidates		LESS				KRIMP	
Dataset	ϵ	*# LE-sets*	*pruning*	$\lvert CT\rvert$	$L(\mathcal{D}, CT)$	*% compr.*	*min-sup*	$\lvert CT\rvert$
adult	2.9	143766	no	**153**	27149706	79.3	1	1941
			yes	**10**	26678350	77.9		1303
course	2.8	455709	no	**918**	1003730	70.6	100	551
			yes	**28**	877284	61.7		285
heart	3.3	414589	no	**115**	117343	87.2	1	108
			yes	**49**	115539	85.8		79
letter recog	3.3	368889	no	**838**	11375860	76.1	50	3395
			yes	**21**	10547561	70.5		1259
mammals	3.8	250628	no	**488**	359116	65.0	200	536
			yes	**30**	314932	57.0		254
mushroom	2.8	437239	no	**241**	5802484	73.5	1	689
			yes	**10**	5474484	69.3		424
pen digits	2.5	71994	no	**160**	4088429	60.5	50	2667
			yes	**28**	3778077	55.9		1091

Table 3: Results of LESS, using the GREEDYCOVER algorithm and a variety of datasets, with comparison to the frequent itemset based method KRIMP. Shown are the threshold ϵ (in bits) for mining low-entropy sets and the number of sets discovered. For LESS, we show for both pruning disabled and enabled, the number of LE-sets selected into the code table, the total compressed size of the data and the achieved compression ratio. For KRIMP we show the minimal support threshold for mining frequent itemsets and the number of selected itemsets.

score was much higher for the real data, even though we directly optimize the compression and not this score. Further, analyzing the contents of the code tables, we also note a significant difference in set cardinality. For the real data, the average set was 2.35 attributes long, while for the randomized data we see elements with an average length of 1.89.

4.4 Reduction and Improvement In Table 3 we present the quantified results of running LESS using the GREEDYCOVER algorithm. The main outcome this table shows is the large reduction in number of low-entropy sets that the algorithm attains. Even for relatively low entropy thresholds, up to 5 orders of magnitude fewer sets are selected. At the same time, the small compressed sizes of the databases show the quality of these descriptions.

We also see that enabling pruning has a strong effect on the number of selected sets: roughly an order of magnitude. Inspection of the code tables shows that the two strategies provide slightly different views on the data. Without pruning, the likelihood maximization process selects more specific sets. Consequently, we see that of the selected sets typically only a few (one or two) very characteristic instantiations find major use. With pruning enabled the process is forced to select more general patterns. This effect is clearly illustrated by the much smaller number of returned sets, of which now multiple instantiations are used often. The much better compression scores show that pruning results in better data descriptions.

Next, we compare the number of patterns our method returns to two other data description methods. First, we compare to KRIMP [19], a lossless approach based on frequent itemsets. Table 3 shows that our method requires far fewer patterns, even although these do describe *all* interactions in the data, instead of just the 1's. It also illustrates that our method is well suited to deal with dense databases, for which the differences grow even larger. For example, at a min-sup of 10% already over 11 million frequent itemsets are mined on *mammals*. Entropy recognises the structure in the data and returns a fraction of this amount in low-entropy sets.

Second, we compare our scores to those of the lossy method proposed by Bringmann and Zimmermann [3] on the largest dataset they considered: *mushroom*. Depending on the selection criterium, their approach returns 21 to 71 itemsets to describe only part of the dataset. Our method, on the other hand, requires only 10 LE-sets to provide a detailed lossless description of the data.

The runtimes of the experiments ranged from one minute up to ten hours. Analysis shows that the runtime is mainly dependent on the number of transactions and particularly the size of the code tables. Hence, the time required for the experiments where pruning keeps the code tables small are in the order of few minutes up to one hour – typically 45 minutes. The experiments with pruning disabled (where the code tables are allowed to grow to hundreds of elements)

typically took up to three hours, with an exception for *mushroom* of ten hours.

4.5 Examining the Code Tables The code tables produced by LESS are small enough for human analysis, for instance through visualization. Figure 1 in Section 1 provides a good example of such a visualization. Each column in the table presents a LE-set with bullets marking the species (rows) that are included in the set. Moreover, as Table 1 shows, one can also zoom in further and investigate the interaction combinations between the variables in full detail.

We observe in Figure 1 that some attributes are included in more than just one set in the code table. At first this looks like redundancy in the description. However, computing a centroid vector for each LE-set in the code table according to the rows it covers and comparing these to the centroids found from the data by the k-means algorithm [14] we notice that the overlapping sets are mostly used in different clusters in the data. For instance the LE-sets $\{$*F. Sylvestris*, *M. Arvalis*, *G. Glis*, *M. Foina*$\}$ and $\{$*F. Sylvestris*, *M. Subter.*, *G. Glis*, *M. Foina*$\}$ are associated to different clusters. Thus, these mappings show that the overlapping sets are not redundant but tuned to describe specific parts of the data.

5 Related Work

Lately, the pattern explosion problem has attracted a lot of research. For frequent pattern mining, lossless methods such as closed [18] and non-derivable [4] itemsets were proposed to remove the redundancy within the pattern set. However, the attained reduction deteriorates heavily under noise. Methods that provide a lossy representation of the complete pattern set include maximal itemsets [2]. Yan et al. [20] proposed a method that selects k representative patterns that together summarize the pattern set well.

Low-entropy sets [10] are a more expressive, entropy-based, generalization of frequent patterns. These allow for more thorough data analysis and reduce the pattern explosion at the same time. However, at high entropy levels the pattern set may still grow prohibitively large. Other related information-theoretic pattern definitions include [12, 17] as well as work on correlated pattern mining [11].

Recently, the approach of finding small subsets of informative patterns has attracted a significant amount of research [3,13,15,19]. Pattern Teams [13] are groups of k non-redundant patterns that have been exhaustively ($k<10$) optimized according to criterions such as joint entropy. Bringmann et al. [3] proposed a greedy variant that can consider larger (100's) pattern sets. Either method is lossy, in the sense that it finds pattern sets that cover only part of the data.

Alternatively, pattern sets can be selected to describe the data best, which falls naturally in the compression approach to data mining [6]. Recently, Siebes et al. [19] introduced the MDL based itemset selection algorithm KRIMP. Although we follow a similar selection approach, the generality and applicability of the methods is rather different. By considering data 0/1–symmetric we can capture all major interactions between attributes, not just co-occurences. Partly thanks to this generalization, LESS yields in the order of tens of patterns, opposed to hundreds to thousands for KRIMP [19]. Through these much smaller numbers inspection by hand is now possible. Also, these pattern sets have a different meaning, as they view the data in terms of strongly interacting variables; not just present items.

Further, the technical solutions we propose are more general. Instead of using ad-hoc order heuristics to determine which patterns describe what part of the data, we introduce a principled way of finding locally optimal covers of the data through the maximum likelihood principle. By using two separate encodings, one identifying the pattern and the other its value instantiation, our framework is more generally applicable. For instance, a promising future research direction would be to expand it to other pattern selection settings where the patterns lack a one-to-one mapping to a specific value, like selecting the most interesting subgroups identified by SQL queries.

6 Discussion

Our novel combination of compression and entropy finds very short, high-quality descriptions of the data. As these descriptions are easily visualized, they can easily be interpreted by humans. They show what is going on in the data, on two levels of detail: providing an overview of the strongly interacting variables in general, and specifying in detail what are the most prominent interactions.

By basing our cover strategies on the maximum likelihood principle, we have a very natural approach to only use instantiations to describe data where this makes sense. Consequently, the code tables capture the significant structure in the data, as the swap-randomization experiments show.

Reconsidering older code table elements once a new LE-set has been admitted increases the quality of the data description even further. Although our method needs to compress the data for each candidate, the measured running times show this approach to be realistic for analysis of large and dense datasets in particular. The candidate set is determined by the max entropy parameter, which may be set as high as is feasible for mining, or makes sense from an analysis point of view. Further, besides the decision of whether or not to prune, there are no parameters: MDL selects the best code table.

LESS combines the best of the lossless and lossy approaches to data description; the number of returned patterns is comparable to the latter, while at the same time our pattern sets do provide a lossless description of the data. Further, these patterns consider both 0's and 1's.

Even though our current implementation is unpolished,

the recorded running times show the method can already realistically be applied for data analysis. However, many possible optimizations are available. One of the most promising would be to just calculate the change the current candidate implies to the previously found best optimum; opposed to calculating a full database cover every time. Speedup on the subset matching could be gained by using a true bitmap representation of the database and the instantiations. Thirdly, paralellization can easily be applied to LESS, both in respect to distributing parts of the database, as well as considering of the candidates distributedly. Using either, or all, these optimizations LESS would become even more applicable for analysis of very large dense databases.

7 Conclusions

We presented LESS, a method for selecting very small collections of highly descriptive low-entropy sets through compression. The small size of these collections facilitates thorough analysis by experts. The interpretability of low-entropy sets makes this analysis even easier. By using entropy instead of frequency, it is particularly suited for mining dense datasets. Further, by regarding data 0/1 symmetric, LESS captures all major interactions in the data, not just co-occurences.

Clearly, entropy is not just defined for binary data, but also for other types, such as real-valued data. Hence, the generalization of this work to such other types of data would make for a both useful and challenging future research direction. Another promising direction would be to apply our framework to other pattern types that lack one-to-one value associations, such as for instance the queries used in subgroup discovery.

Acknowledgments

Jilles Vreeken is supported by the NWO Computational Life Sciences Programme. Hannes Heikinheimo is partially supported by a grant from the Nokia Foundation.

References

[1] R. Agrawal, H. Mannila, R. Srikant, H. Toivonen, and A. I. Verkamo. Fast discovery of association rules. In *Advances in Knowledge Discovery and Data Mining*, pages 307–328. AAAI, 1996.

[2] R. J. Bayardo. Efficiently mining long patterns from databases. In *ACM SIGMOD Conference on Management of Data*, pages 85–93, 1998.

[3] B. Bringmann and A. Zimmermann. The chosen few: On identifying valuable patterns. In *IEEE International Conference on Data Mining*, pages 63–72, 2007.

[4] T. Calders and B. Goethals. Mining all non-derivable frequent itemsets. In *Proceedings of the 6th European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases*, pages 74–85, 2002.

[5] F. Coenen. The LUCS-KDD discretised/normalised ARM and CARM data library. 2003.

[6] C. Faloutsos and V. Megalooikonomou. On data mining, compression and kolmogorov complexity. In *Data Mining and Knowledge Discovery*, volume 15, pages 3–20. Springer, 2007.

[7] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.

[8] A. Gionis, H. Mannila, T. Mielikäinen, and P. Tsaparas. Assessing data mining results via swap randomization. In *ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pages 167–176, 2006.

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

[10] H. Heikinheimo, E. Hinkkanen, H. Mannila, T. Mielikäinen, and J. K. Seppänen. Finding low-entropy sets and trees from binary data. In *ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pages 350–359, 2007.

[11] Y. Ke, J. Cheng, and W. Ng. Mining quantitative correlated patterns using an information-theoretic approach. In *ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pages 227–236, 2006.

[12] A. J. Knobbe and E. K. Y. Ho. Maximally informative k-itemsets and their efficient discovery. In *ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pages 237–244, 2006.

[13] A. J. Knobbe and E. K. Y. Ho. Pattern teams. In *European Conference on Principles and Practice of Knowledge Discovery in Databases*, pages 577–584, 2006.

[14] J. MacQueen. Some methods for classification and analysis of multivariate observations. In *Proceedings of the 5th Symposium on Mathematical Statistics and Probability*, 1967.

[15] T. Mielikäinen and H. Mannila. The pattern ordering problem. In *European Conference on Principles and Practice of Knowledge Discovery in Databases*, pages 327–338, 2003.

[16] A. J. Mitchell-Jones, G. Amori, W. Bogdanowicz, B. Krystufek, P. J. H. Reijnders, F. Spitzenberger, M. Stubbe, J. B. M. Thissen, V. Vohralik, and J. Zima. *The Atlas of European Mammals*. Academic Press, 1999.

[17] S. Morishita and J. Sese. Traversing itemset lattice with statistical metric pruning. In *PODS*, pages 226–236, 2000.

[18] N. Pasquier, Y. Bastide, R. Taouil, and L. Lakhal. Discovering frequent closed itemsets for association rules. *Lecture Notes in Computer Science*, 1540:398–416, 1999.

[19] A. Siebes, J. Vreeken, and M. van Leeuwen. Item sets that compress. In *SIAM Conference on Data Mining*, pages 393–404, 2006.

[20] X. Yan, H. Cheng, J. Han, and D. Xin. Summarizing itemset patterns: A profile-based approach. In *ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pages 314–323, 2005.