VOUW: Spatial Pattern Mining using the MDL Principle

Micky Faas and Matthijs van Leeuwen

Leiden Institute for Advances Computer Science

Abstract

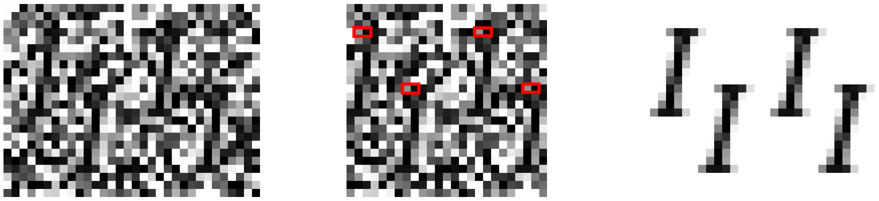
- MDL is a principle that utilizes compression as a means of describing data
- There exist many MDL-based algorithms for a variety of problems
- Few (no?) solutions exist to mine grid-like data based on MDL
- VOUW is a novel approach to discover patterns and structural relations in 2D, discrete datasets
- We propose both a theoretical framework as well as a complete, optimized implementation

1 Introduction

In recent years, an emerging class of algorithms utilizing the *Minimum Description Length (MDL) principle* [5, 2] have become more and more common in the field of explanatory data analysis. Examples of such approaches include the early Krimp [7] or the more recent Classy [4] algorithms. The MDL principle was first described by Rissanen in 1987 [5] as a practical implementation of Kolmogorov Complexity [3]. Central to MDL is the notion that 'learning' can be thought of as 'finding regularity' and that regularity itself is a property of data that is exploited by *compressing* said data. Therefore by compressing a dataset, we actually learn its structure — how regular it is, where this regularity occurs, what it looks like — at the same time. Indeed MDL postulates that the most optimal compression (minimal description) of a given dataset provides the best description of that data.

The problem that MDL tries to solve first and foremost, is that of *model selection*: given a multitude of explanations (models), select the one that fits the data best. In addition to this, MDL has also been demonstrated to be very effective in materialization of a specific model given the data. In this case, the model is predetermined and we want to find the parameters to fit the data. A similar problem class is solved in pattern mining: here the 'parameters' are the discrete building blocks that make up patterns in the data. In fact, the Krimp algorithm mentioned earlier solves a specific subclass of the problem. In this paper we will look at another class of pattern mining problems that is rarely addressed in literature, namely that of spatial structure discovery.

Let us demonstrate the problem of spatial pattern mining by giving a brief example. Figure 1a shows a grayscale image that we assume is random 'white



(a) 32 × 24 noise 'matrix'

(b) Pair $(146, 11)$

(c) Pattern 'I' occurs four times.

Fig. 1: Brief example of spatial pattern mining

noise'. For convenience, we will interpret the image as a 32×24 matrix with values on the interval $[0; 255]$. If we were to look at all horizontal pairs of elements, we would find that the pair $(146, 11)$ is, among others, statistically more prevalent than the initial assumption of random would suggest. Figure 1b highlights the locations where these pairs occur: we have just discovered the first repeating structure in our dataset! If we would continue to try all combinations of elements that 'stand out' from the background noise, Figure 1c shows that we will eventually find that the matrix contains four copies of the letter 'I' set in 16 point Garamond Italic.

The 35 elements that make up a single 'I' in the example, are said to form a *pattern*. We can use this pattern to describe the matrix in an other way than '768 unrelated values'. For example, we could describe it as 628 unrelated values plus pattern 'I' at locations $(5, 4), (11, 11), (20, 3), (25, 10)$, separating the structure from the accidental (noise) data. Since this requires less storage space than before, we have also compressed the original data. See how at the same time we have learned something about the data: we did not know about the 'hidden I's before.

1.1 Spatial pattern mining

As the example above very roughly demonstrates, spatial pattern mining is the problem of finding recurring (local) structure in multi-dimensional matrices of data. It is different from graph mining, as a matrix is more rigid and each element has a fixed degree of connectedness/adjacency. It is also unrelated to linear algebra, other then using the term 'matrix' and a comparable style of notation. Furthermore in this context, matrix elements can only be discrete, rows and columns have a fixed ordering and the semantics of a value is position-independent.

The problem of spatial pattern mining can roughly be divided into three classes. The first class consists of three subclasses: identical recurring structures in (1a) an otherwise empty (sparse) matrix, (1b) differently distributed noise and (1c) similarly distributed noise. The second class contains the same subclasses but adds that the recurring structures can also be overlaid with noise and are therefore not identical. The third class is also a continuation of the first class and requires that the recurring structures are identical after some optional transformation (such as mirror, inverse, rotate, etc.). These classes also represent an increasing difficulty level and serves as a rough benchmark for the performance of an algorithm.

Although we could intuitively solve the example, in reality it is much more complex. Is the 'I' the best pattern we can find (and why), are there any more patterns and, most important, how do we find it (quickly) in an unknown and/or much larger dataset? After formally defining these problems and their contexts we will introduce VOUW, a novel spatial pattern mining algorithm based on the MDL principle.

2 Related Work

Krimp [7] is probably one of the first explanatory data mining approaches using MDL and also one of the sources of inspiration for this paper. Since then, many papers on this topic have been published, such as Slim **??** (frequent item set mining, like Krimp), Classy **??** (mining classifiers), *TODO: list more*

The work by Campana et al. [1] also uses matrix-like input data (textures) and develops a similarity measure based on MDL. Their method, however, cannot be used for *explanatory* data analysis as they use a generic image compression algorithm that is essentially a black box.

This is still far from complete

3 Theoretical Framework

3.1 On Patterns and Matrices

In this subsection we will introduce a method for describing and decomposing tabular or matrix-like data. It is this concise notation that allows us to reason about what one can and cannot expect to find from a set of data and what this data could look like. In later subsections we will use these tools to gradually connect to the MDL principle and finally to a practical search algorithm.

We define spatial pattern mining on bounded, discrete and two-dimensional geometric data. We represent this data as an $M \times N$ matrix A whose rows and columns are finite and in a fixed ordering (i.e. reordering rows and columns semantically alters the matrix). Elements $a_{i,j}$, where row i is on $[0; N)$ and column j is on $[0; M)$, holds that $a_{i,j} \in S$, the finite set of symbols occurring in A. We will denote A as a matrix because of the convenience of the linear algebra notation[1]

MDL tells us to search for the shortest (optimal) description of A. This description then tells us everything there is to know about A in the most succinct way possible. The principle behind MDL is that we can approximate an optimal

[1] In any practical application however, A will not represent a system of linear equations but rather hold some form of sampled data that can be placed on a grid.

$$A = \begin{bmatrix} 1 & \cdot & \cdot & \cdot & \cdot & 1 \\ \cdot & 1 & \cdot & \cdot & \cdot & \cdot \\ 1 & \cdot & \cdot & \cdot & 1 & \cdot \\ \cdot & 1 & \cdot & \cdot & \cdot & 1 \\ 1 & \cdot & 1 & \cdot & \cdot & \cdot \\ \cdot & 1 & \cdot & 1 & \cdot & \cdot \end{bmatrix},$$

$$\bar{H} = \begin{bmatrix} X & \cdot & \cdot & \cdot & \cdot & Y \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ X & \cdot & \cdot & \cdot & X & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ X & \cdot & X & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \end{bmatrix},$$

$$H = \{X = \begin{bmatrix} 1 & \cdot \\ \cdot & 1 \end{bmatrix}, Y = \begin{bmatrix} 1 \end{bmatrix}\}$$

Fig. 2: Example decomposition of A into instantiation $\bar{H}$ and patterns X, Y

description through the compression of the original data. This optimal description, let us call it A', is only optimal if we can unambiguously reconstruct A from it and nothing more — the compression is both minimal and 'lossless'. Intuitively compression means that we would like to have a way to just define A using as few building blocks as possible. We illustrate this using the example in Figure 2. Given the matrix A we try to decompose it in recurring substructures which we will call *patterns*, denoted X and Y in the example. At any given time, the set of patterns which we call the *model* of a A, is denoted H_A. Given only these patterns, A cannot be reconstructed because we need a mapping from the model H_A back to A. This mapping represents in what MDL calls the *matrix A given the model* H_A. In statistics this is often the structural versus the accidental information, but in this context we think of it merely as 'instructions' of how to reconstruct A. Let us call the set of all instructions required to rebuild A from H_A the *instantiation* of H_A. It is denoted by $\bar{H}_A$ in the example. Notice how this instantiation essentially tells us where in A each pattern from H_A was originally located. The result is a notation that allows us to express matrix A as if decomposed into sets of local and global spatial information.

Now that we have this top-down concept of the decomposition of A, we will continue to describe its constituents in more detail.

Patterns and Instances Intuitively we can think of a pattern as some submatrix X of the original matrix A. This submatrix is not necessarily complete (elements may be $\cdot$, the empty element), which gives us the ability to precisely cut-out any irregular-shaped part of A. We additionally require the elements of X to be adjacent (horizontal, vertical or diagonal) to at least one non-empty element. While this limits the amount of possible patterns somewhat, it will later on also reduce the computational effort dramatically. We will now define a pattern to be the smallest submatrix to completely contain all elements of X.

Definition 1. *We define **pattern** X as an $M_X \times N_X$ submatrix of A where any non-empty element may optionally be replaced by $\cdot$ such that:*

- *Any non-empty element in X is adjacent (horizontal, vertical or diagonal) to at least one other non-empty element.*
- *The first and last rows and columns contain at least one non-empty element.*

From this definition we see that the dimensions $M_X \times N_X$ give essentially the smallest rectangle around X (the *bounding box)*. As a more useful measure we therefore also define the cardinality $|X|$ of X as the number of non-empty elements. We call a pattern X with $|X| = 1$ a *singleton pattern*, i.e. a pattern containing exactly one element of A. [2]

One element in each pattern is given the special function of *pivot*.

Definition 2. *The **pivot** $p(X)$ of pattern X is the first non-empty element of X in the first non-empty column of the first (non-empty) row of X.*

A pivot can be thought of as a fixed point in a pattern X which we can use to position its elements in relation to A. The precise translation we apply to a particular pattern we call an *offset*. An offset is a tuple $\delta = (i, j)$ that is on the same domain as an index in A. We realize this translation by placing all elements of X on an empty $M \times X$ size matrix in such way that the pivot element is at (i, j). We formalize this concept with the **instantiation operator** $\oplus$.

Definition 3. *In the context of an $M \times N$ matrix A, we define the **instance** $X \oplus \delta$ as the incomplete $M \times N$ matrix containing all elements of X such that the pivot $p(X)$ is at index (i, j) and the distances between all original elements are preserved. The resulting matrix contains no additional non-empty elements.*

So according to this definition $\oplus$ adds 'padding' around the elements of a pattern to align its pivot to a certain offset (i, j). Obviously this does not yield a valid result for an arbitrary offset (i, j). We want to limit ourselves to the space of pattern instances that are actually valid in relation to matrix A. Therefore two simple constraints are needed: (1) an instance must be *well-defined*: placing pivot $p(X)$ is at index (i, j) results in a $M \times N$ matrix that contains all elements of X, and (2) elements of instances cannot overlap, meaning each element of A should be described at most once. This allows for a description that is both unambiguous and minimal.

Definition 4. *Two pattern instances $X \oplus \delta_x$ and $Y \oplus \delta_y$, with $\delta_x \neq \delta_y$ are **non-overlapping** if $|(X \oplus \delta_x) + (Y \oplus \delta_y)| = |X| + |Y|$.*

[2] We will often slightly abuse notation by using single-index elements or using set notation for any matrix. For instance, when we write $x_i \in X$, we mean that x_i is the i-th non-empty element in X with $1 \leq i \leq |X|$. By convention we will always use row-major ordering in these cases.

The definitions for patterns and their instances now give the appropriate tools to describe a mapping from sets of patterns to the original matrix A. Say we have a set of patterns H over A, we would like to have a set of 'instructions' of where instances of each pattern should be positioned in order to obtain A. When we look at the example in Figure 2, we see that we could use again an $M \times N$ matrix for this. This matrix contains elements that are in H such that each index corresponds to the offset of that specific pattern's instance.

Definition 5. *Given the set of patterns H, the* ***instantiation (matrix)*** $\bar{H}$ *is an incomplete $M \times N$ matrix with the set of possible elements being H, i.e. $\bar{h}_{i,j} \in H$ for all (i, j). For all non-empty elements $\bar{h}_{i,j}$ it holds that $\bar{h}_{i,j} \oplus (i, j)$ is an instance of $\bar{h}_{i,j}$ in A that is not non-overlapping with any other instance $\bar{h}_{i',j'} \oplus (i', j')$.*

The above definition creates the interesting proposition that the offset to each instance is unique. Given that a pattern's pivot is placed exactly at one offset and that instances must be non-overlapping, makes this indeed believable. Later when we inductively define the set $\bar{H}$ of all instantiation matrices, this will be shown to be true more formally.

Configurations In the previous subsections we silently omitted an important constraint in instantiating patterns. An instance has only meaning in the context of matrix A if their respective elements match. In order words, if an instance $X \oplus (i, j)$ has all its non-empty elements identical to the corresponding indices in A, this means that pattern X matches A in (i, j). In this case the match is exact, formally

Definition 6. *The instance $\bar{X} = X \oplus \delta$ is an* ***exact match on*** *A if for all non-empty $\bar{x}_{i,j} \in \bar{X}$ it holds that $\bar{x}_{i,j} = a_{i,j}$.*

While this form of straightforward matching can be desirable in some cases, it may lead to a bloated model in others. Take the example from Figure 3 that lists matrix A and four patterns. Pattern W is obviously an exact match to five of the six elements of A. Pattern X, while not an exact match, is also a good candidate to describe A although it is multiplied by a constant factor of two. Pattern Y is a near-exact match: only one element is off - this could be due to noise for example. Pattern Z is no match for the values in A, but notice that it is identical in structure to the other patterns: it places as many values at the same indices.

The example above suggests that we could do one more step of decomposition: just like we decomposed the original matrix into global structure (instantiation) and local structure (pattern), we decompose patterns into structure and magnitude.

Definition 7. *The* ***configuration*** *$S(X)$ of pattern X is an equally-sized Boolean matrix such that $S(X)_{i,j} = 1$ whenever $X_{i,j}$ is non-empty.*
The ***magnitude*** *$M(X)$ of pattern X is the string of values obtained by taking each non-empty element of X in row-major order.*

$$A = \begin{bmatrix} 1 & 2 \\ \cdot & 1 \\ 1 & 2 \end{bmatrix} \middle| W = \begin{bmatrix} 1 & 2 \\ \cdot & 1 \\ 1 & 2 \end{bmatrix},$$

$$X = \begin{bmatrix} 2 & 4 \\ \cdot & 2 \\ 2 & 4 \end{bmatrix}, Y = \begin{bmatrix} 1 & 2 \\ \cdot & 1 \\ 1 & 1 \end{bmatrix}, Z = \begin{bmatrix} 2 & 2 \\ \cdot & 0 \\ 4 & 3 \end{bmatrix},$$

Fig. 3: Examples of structurally equivalent patterns with varying degrees of similarity.

In the example we can see that $S(X) = \left[\begin{smallmatrix} 1 & 1 \\ \cdot & 1 \\ 1 & 1 \end{smallmatrix}\right]$ and that $M(X) = 1\ 2\ 1\ 1\ 2$. In fact, because all four patterns have the same configuration, we say that they are *isomorphic*. As such we write $X \cong Y$ iff $S(X) = S(Y)$.

To understand the importance of separating the structure and magnitude of patterns, let us briefly expand the example above. Suppose we have a large matrix B that consists of n clusters of matrices W and X from Figure 3. To determine the optimal description of B, we might want to look at the prevalence of each submatrix. Say it contains $\frac{n}{2}$ W's and $\frac{n}{2}$ X's. In this case it makes sense to include both patterns in the description. However, we could also exploit the fact that the structure of both patterns is equivalent and make the description more concise by only storing $S(W)$ and then $M(W)$ and $M(X)$ separately. Now imagine that B only contains one W and $n-1$ X's. In this case the one W might be an anomaly that we would like to detect. However, it could also be due to noise in the data in which case we would like to describe B just using n X's. It is impossible to make this distinction beforehand.

One possibility for solving this problem is to let the MDL equation decide whether the stray W is an anomaly or not. Recall that according to the MDL principle the most succinct description is the best. Therefore if the amount of 'effort' required to transform X into W is small, we should probably encode that one W using X. In that case it is considered noise, while it is probably an anomaly if doing so would yield a larger description.

3.2 The Problem and its Solution Space

Constructing Patterns As patterns are the building blocks of our model, we also need to have a practical way to construct them. We will do this by joining smaller patterns together to create increasingly large patterns and, as we will later see, this concept also forms the basis of the search algorithm. However, two patterns could be joined in any different number of ways. We can define the exact way two patterns should be joined by enumerating the distance of their respective pivots. In order to this we, we use pattern instances as an intermediate step.

Recall that instances are simply patterns projected on an $M \times N$ matrix, containing the same elements as the patterns at their original distances. This makes $\oplus$ trivially reversible by removing all completely empty rows and columns.

Definition 8. *Let $X \oplus \delta$ be an instance of X, then by definition we say that $\ominus(X \oplus \delta) = X$.*

We can now define a sum on two instances to get a new instance that combines both operands. Even though the result is not a pattern, we can always obtain it by using $\ominus$. The example in Figure 4 illustrates this process. We start by instantiating X and Y with offsets $(1, 0)$ and $(1, 1)$ respectively. This yields the non-overlapping $\bar{X}$ and $\bar{Y}$, which we simply add up to obtain $\bar{Z}$. The new pattern contained in $\bar{Z}$ can be easily identified by removing the top empty row. We formally describe this mechanism in Theorem 1. As we will see later, this will become the fundamental operation of our algorithm.

$$X = \begin{bmatrix} 1 & \cdot \\ \cdot & 1 \end{bmatrix}, Y = \begin{bmatrix} 1 \end{bmatrix},$$

$$\bar{X} = X \oplus (1, 0) = \begin{bmatrix} \cdot & \cdot \\ 1 & \cdot \\ \cdot & 1 \end{bmatrix},$$

$$\bar{Y} = Y \oplus (1, 1) = \begin{bmatrix} \cdot & \cdot \\ \cdot & 1 \\ \cdot & \cdot \end{bmatrix},$$

$$\bar{X} + \bar{Y} = \begin{bmatrix} \cdot & \cdot \\ 1 & 1 \\ \cdot & 1 \end{bmatrix},$$

$$Z = \ominus(\bar{X} + \bar{Y}) = \begin{bmatrix} 1 & 1 \\ \cdot & 1 \end{bmatrix}$$

Fig. 4: Example of Theorem 1. The 2×3 input matrix is not shown.

Theorem 1. *Given two non-overlapping instances $\bar{X} = X \oplus \delta_X$ and $\bar{Y} = Y \oplus \delta_Y$, the sum of the matrices $\bar{X} + \bar{Y}$ is another instance. We observe that pattern $Z = \ominus(\bar{X} + \bar{Y})$ such that $\bar{X} + \bar{Y} = Z \oplus \delta_X$.*

Notice that this sum has the limitation that two instances can only be summed if they do not overlap. While this is a serious limitation, we will show in the next subsection that it is not of any practical relevance.

The Set $\mathcal{H}_A$ and $\bar{\mathcal{H}}_A$ In the previous subsections we have given a means to describe a matrix A in a different way, namely by means of patterns and instances. If we succeed in describing A, using our notation, in a more concise way than just A itself, we have learned something about the local and global structure of A and perhaps even about anomalies or noisy values. In this context, we see a clear relation the MDL principle and learning.

In order to find a short(er) description, we will first have to define our search space and the way solutions are to be constructed. We begin by defining the *model class* $\mathcal{H}$, the set of all possible models for all possible inputs. Without any prior knowledge, this is the search space of our algorithm. We will first look at a more bounded subset of $\mathcal{H}$, namely the set $\mathcal{H}_A$ of all possible models for A, and its counterpart, the set $\bar{\mathcal{H}}_A$ of all possible instantiations to these models. We will also take H_A^0 to be the model with only singleton patterns (patterns of length 1). As singletons are just individual elements of A, we can simply say that $H_A^0 = S$. The instantiation matrix corresponding to H_A^0 is denoted $\bar{H}_A^0$. Given that each element of this matrix must correspond to exactly one element of A in H_A^0, we see that each $\bar{h}_{i,j} = a_{i,j}$ and so $\bar{H}_A^0$ is equal to A.

Using H_A^0 and $\bar{H}_A^0$ as base cases we can now inductively define the set $\bar{\mathcal{H}}_A$ of all instantiations of all models over A:
Base case: $\bar{H}_A^0 \in \bar{\mathcal{H}}_A$
By induction: If $\bar{H}$ is in $\bar{\mathcal{H}}_A$ then take any pair $\bar{h}_{i,j}, \bar{h}_{k,l} \in \bar{H}$ such that $(i,j) \leq (k,l)$ in lexicographical order. The set $\bar{H}'$ is also in $\bar{\mathcal{H}}_A$ for $\bar{H}' = \bar{H}$ and

$$\begin{aligned} \bar{h}'_{i,j} &:= \ominus\big(\bar{h}_{i,j} \oplus (i,j) + \bar{h}_{k,l} \oplus (k,l)\big) \\ \bar{h}'_{k,l} &:= \cdot \end{aligned}$$

In this definition of $\bar{H}_A^0$ we inductively replace two instances with their sum. Indeed we can add any two instances together in any order and eventually this results in just one big instance that is equal to A. The elegance in this is that, by this inductive definition, the instances never overlap and thus their sum is always a valid instance on A. Note that when we take two elements $\bar{h}_{i,j}, \bar{h}_{k,l} \in \bar{H}$ we force $(i,j) \leq (k,l)$ to be in lexicographical order, so that know the pivot of the new pattern to coincides with $\bar{h}_{i,j}$. We can then leave $\bar{h}_{k,l}$ empty.

The construction of instantiation matrices also implicitly defines the corresponding models. While this may seem odd — defining models for instantiations instead of the other way around — note that there is no unambiguous way to find an instantiation matrix for a given model[3]. Instead we find the following trivial definition by applying the inductive construction rule above.

Definition 9. *The set $\mathcal{H}_A$ of all models over A is given by*

$$\mathcal{H}_A = \Big\{ \big\{ \ominus(\bar{h}) \mid \bar{h} \in \bar{H} big \big\} \Bigm| \bar{H} \in \bar{\mathcal{H}}_A \Big\}.$$

So for any instantiation $\bar{H} \in \bar{\mathcal{H}}_A$ there is a corresponding set in $\mathcal{H}_A$ of all patterns that occur in $\bar{H}$. This results in an interesting symbiosis between model and instantiation: increasing the complexity of one decreases that of the other. When plotting this construction a tightly connected lattice appears such as that of Figure 5.

The Optimal Model The final step in the process of describing A is to select the 'best' or 'most optimal' model from the set of $\mathcal{H}_A$. While intuitively we can

[3] Notice that this is a problem for any real-world implementation. We will describe a heuristic to derive instantiation matrices from model and data in the next section.

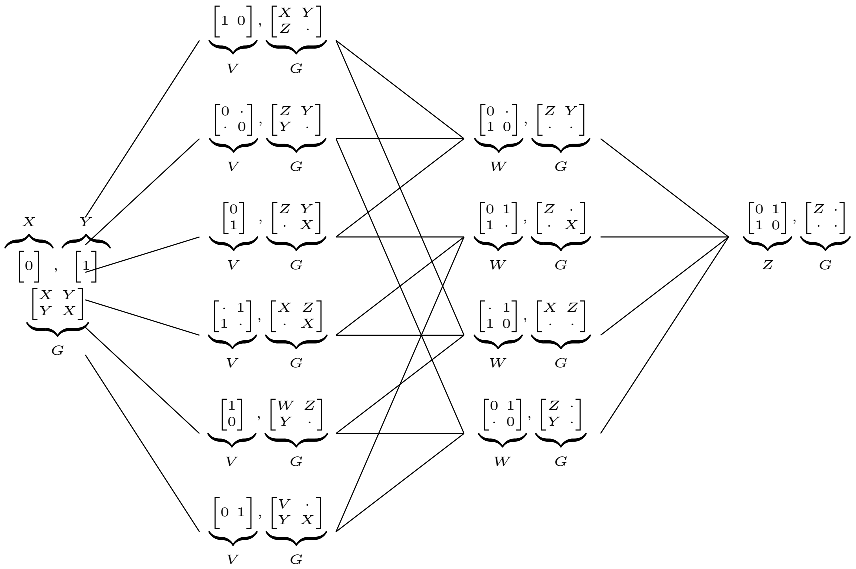

X
Y
[0] , [1]
[X Y / Y X]
G
[1 0] , [X Y / Z ·]
V G
[0 · / · 0] , [Z Y / Y ·]
V G
[0 / 1] , [Z Y / · X]
V G
[· 1 / 1 ·] , [X Z / · X]
V G
[1 / 0] , [W Z / Y ·]
V G
[0 1] , [V · / Y X]
V G
[0 · / 1 0] , [Z Y / · ·]
W G
[0 1 / 1 ·] , [Z · / · X]
W G
[· 1 / 1 0] , [X Z / · ·]
W G
[0 1 / · 0] , [Z · / Y ·]
W G
[0 1 / 1 0] , [Z · / · ·]
Z G

Fig. 5: The model space lattice for a 2×2 Boolean matrix. Here X and Y are the initial singleton patterns and G is the instance matrix. V and W are intermediate patterns and Z is the final, completely over fit, pattern.

understand that not every model we pick is equally fitting in terms of how well it describes A, this concepts needs to be formalized before we can even begin to search $\mathcal{H}_A$. To this end we make the connection with the MDL principle that says, informally, that the most concise encoding of the data gives us the best description. Since we are using two-part MDL, it is very convenient that we have already split the problem into two parts: a model H_A and an instantiation $\bar{H}_A$. Together they form A' and according to MDL, their sum must be minimized. The two-part MDL equation looks like:

$$L_1(H_A) + L_2(A|H_A)$$

Here the functions L_1, L_2 are two independent length functions that make up the coding scheme for two-part MDL. This minimization is often thought of as compression, although we will not actually write any encoded data. This leaves us with the task of finding an encoding scheme that encodes both model and instantiations lossless and without redundancy.

Say we have some set of code words $\{C_0, C_1, \ldots, C_n\}$. These symbols could be for example be a code word for each pattern $X_0, X_1, \ldots, X_n$ in a model. We now want to find optimal lengths $l_0, l_1, \ldots, l_n$ to assign to each code word using the Kraft-inequality such that holds that $\sum_{i=0}^{N} r^{-l_i} \leq 1$. In this case we say that $r = 2$ (symbols 0 and 1), so we can measure code lengths in bits. The Kraft-inequality gives us a bijection between code lengths and probability distributions. This is one of the main ideas of Shannon's entropy, which plays also an important role in the MDL principle. In our example we can write a probability distribution $P(X)$ for $X \in H_A$, as the probability of pattern X occurring in our instance set. Given these probabilities we can use $l_i = -\log(P(X_i))$ to compute the exact number of bits a pattern should optimally be encoded with[4].

3.3 Encoding Models and Instantiations

To be able to compute the length of a given code word we must know the probability of that word occurring in our data. This information must also be available to the hypothetical decoder as otherwise the encoding is not lossless. Sometimes this is not practical as we do not know the probability distribution beforehand. For example, given an arbitrary encoded A', we do not know the probability that each pattern occurs in the instantiation matrix. We could also encode this information and pass it to the decoder separately, but this is generally a bad idea. It cannot be stressed enough that the leanest encoding gives us the most information about the true compression ratio we achieve, while bookkeeping and meta-information only incur an undue bias.

Instead of using the optimal distribution $L(p) = -\log(p)$ we can also use a different distribution, as long as the encoder and hypothetical decoder agree upon which distribution is used. Such distribution is often called a *prior* and is used to fix 'prior knowledge' that does not have to be encoded explicitly. Although the optimal distribution can only be approached, it is sometimes a justifiable trade-off if it means that we do not have to encode uninteresting information.

A good example of a prior that we will be using is the universal code for integers [**?**]. The corresponding length function $L_{\mathbb{N}}(n)$ gives the number of bits required to encode an arbitrary n and is defined as $L_{\mathbb{N}}(n) = \log *(n) + \log(c_0)$ with $\log *(n) = \log(n) + \log\log(n) + \cdots$ and $c_0 = 2.865064$ to satisfy the Kraft-inequality. This code is obviously not uniform and assigns a longer code to a larger n. We will use this code to encode arbitrary integers.

To encode the instantiation matrix we will use the *prequential plug-in code* [**?**]. The prequential plug-in code is defined for sequences of one item at a time and updates the probability of each item as it is encoded, such that the probability need not be known in advance. It has the favourable property of being asymptotically equal to the optimal code for large sequences. Say we want to encode all elements $\bar{h}_i \in \bar{H}$ lexicographical order, we define:

Definition 10.

$$P_{plugin}(y_i = \bar{h}_i \mid y^{i-1}) = \frac{|\{y \in y^{i-1} \mid y = \bar{h}_i\}| + \epsilon}{\sum_{X \in H} |\{y \in y^{i-1} \mid y = X\}| + \epsilon}$$

[4] Notice how common code words are shorter then rarely used code words. According to the Kraft inequality this gives us an optimal code. The number of bits we compute are real numbers and not integers. While this does certainly not result in a practical encoding, for the purpose of model selection we do not actually need to encode the data as we are only interested in its hypothetical length.

Here y_i is the i-th element to be encoded and y^{i-1} is the sequence of elements encoded so far. We initialize the base case (no element has been sent yet) with a pseudocount ϵ, which gives $P_{plugin}(y_1 = \bar{h} \mid y^0) = \frac{\epsilon}{\epsilon|H|}$. We pick $\epsilon = 0.5$ as it is used generally with good results.

Let us adapt this principle to the problem of encoding patterns. The first step here is to determine the probability that each unique element (instance of a pattern) in $\bar{H}$ occurs.

Definition 11. *Given a set of instantiations $\bar{H}$, we define $U(X)$ as the* ***usage*** *of pattern X such that*

$$U(X) = |\{\bar{h}_i \in \bar{H} \mid \bar{h}_i = X\}|.$$

From this definition we see that the *usage* of a pattern is a sum of how often it occurs as an instance. We can use this function to simplify things a little by realizing that we actually know the precise number of instances per pattern on the side of the decoder, but not as the decoder. This information can be used to slightly rephrase Definition 10 to be able to encode items in arbitrary order. This produces the length function of the instantiation matrix $\bar{H}$ as follows[5]:

$$\begin{aligned}
L_{pp}(\bar{H} \mid P_{plugin}) &= \sum_{i=1}^{|\bar{H}|} -\log \frac{|\{y \in y^{i-1} \mid y = \bar{h}_i\}| + \epsilon}{\sum_{X \in H} |\{y \in y^{i-1} \mid y = X\}| + \epsilon} \\
&= \sum_{X_i \in h}^{|H|} -\log \prod_{j=0}^{U(X_i)-1} \frac{j + \epsilon}{\sum_{k=1}^{i-1} U(X_k) + j + \epsilon|H|} \\
&= -\log \frac{\prod^{X_i \in H} \prod_{j=0}^{U(X_i)} j + \epsilon}{\prod_{j=0}^{|\bar{H}|-1} j + \epsilon|H|} \\
&= -\sum_{X_i \in h}^{|H|} \left[\log \frac{\Gamma(U(X_i) + \epsilon)}{\Gamma(\epsilon)} \right] + \log \frac{\Gamma(|\bar{H}| + \epsilon|H|)}{\Gamma(\epsilon|H|)}
\end{aligned}$$

Lastly, in addition to $L_{\mathbb{N}}$ and L_{pp} we also define the length of the uniform distribution $L_0(n) = log(n)$. That is, when n items have equal probability they all receive a code of equal length $log(n)$.

The length function for incomplete matrices. To losslessly encode A' we have to encode both H and $\bar{H}$ individually. Recall that both instantiations and patterns are both matrices. It is therefore tempting to utilize the same length function for both. Empirical evidence has shown that this is not a good idea though and the main reason for this is the fact that prequential plug-in code

[5] Here we use the fact that we can interchange sums of logarithms with logarithms of products and that those terms can be moved around freely. Moreover we convert the real-valued product sequences to the Gamma function Γ, which is the factorial function extended to real and complex numbers such that $\Gamma(n) = (n-1)!$.

behaves different for small sequences (patterns) than it does for large sequences (the instantiation matrix). Furthermore, we do not consider certain values such as the size of the instantiation matrix because it is constant, however, the size of each individual pattern is not. We therefore have to construct a different length function for each type of matrix. These are listed in Table 1.

	Matrix	Bounds	ne Elements	Positions	Symbols
$L_p(X)$	Pattern	$L_0(MN)$	$L_{\mathbb{N}}(\binom{M_X N_X}{\lvert X \rvert})$		$L_0(\lvert S \rvert)$
$L_1(H)$	Model	N/A	$L_N(\lvert H \rvert)$	N/A	$L_p(X \in H)$
$L_2(\bar{H})$	Inst. mat.	*constant*	$L_0(MN)$	*implicit*	$L_{pp}(\bar{H})$
$L_3(E)$	Error mat.	*constant*	$L_0(MN)$	$L_0(MN)$	$L_0(\lvert S \rvert)$

Table 1: Length computation for the different classes of matrices. The total length is the sum of the listed terms. This table also lists a length function for the error matrix that will be discussed shortly hereafter.

Each length function has four (optional) terms. First we encode the total size of the matrix. Since we assume MN to be known/constant, we can use this constant to define the distribution $L_0(MN)$. This term encodes an arbitrary index of A with equal lengths for each index. Next we encode the number of elements that are non-empty. Notice how for patterns, this value is encoded together with the third term, namely the positions of the non-empty elements. Because we have encoded $M_X N_X$ in the first term, we may now use it as a constant. We use it in the binominal function to enumerate the number of ways we can place the non-empty elements ($|X|$) onto a grid of $M_X N_X$. This gives us both *how many* non-empties there are as well as *where* they are. Finally the fourth term is the length of the actual symbols that encode the elements of matrix. In case we encode single elements of A, we simply assume that each unique value in A has an equal possibility of occurring. For the instantiation matrix, which encodes symbols to patterns, the prequential code is used as demonstrated before.

Configurations and the error matrix TODO

Gain Given the length function described previously, we can now quantify how well a certain candidate model compresses and thus how much information it reveals about the original data. It is trivial to define the amount of benefit that is to be gained by transforming model, instantiation matrix and error matrix $H, \bar{H}, E$ to a different $H', \bar{H}', E'$.

Definition 12. *The function* gain : $H, \bar{H}, E \mapsto H', \bar{H}', E'$ *is defined as*

$$\Big(L_1(H') + L_2(\bar{H}') + L_3(E')\Big) - \Big(L_1(H) + L_2(\bar{H}) + L_3(E)\Big)$$

4 A Search Algorithm

I have this partially, but so much has changed that it will need to be rewritten anyway. I would like to split it in two parts: the absolute minimal baseline algorithm in one subsection, and then another to describe all the extensions and optimizations.

4.1 An Outline

Given an input matrix, the algorithm is initialized with: (a) probability mass function of the values in the matrix, (b) singleton pattern for each unique element in the matrix and (c) the under-fit instantiation matrix containing only singletons. The baseline algorithm is composed out of four stages that are repeated every iteration:

1. Candidate search
2. Gain computation
3. Candidate selection
4. Merging of patterns and their instances

Candidate Search

Gain Computation

Selecting and Merging Candidates

4.2 Extensions and Improvements

Pre-processing

Reducing Self-Overlap

Alternative Heuristics

Local Search

5 Experiments and Results

Outline of possible experiments:

- Quality of the pattern set.
  - Optimal vs heuristics (not practical, scaling/generalization issues)
  - Synthetic (pre-generated models generate data, fun, let's program that)
  - Compression ratio (apparently there is a theoretical basis on why these metrics are representative)
- Baseline vs optimizations (will optimizations alter quality of the outcome?)
- Anecdotal evidence (examples, demonstration)

6 Discussion

7 Conclusion

References

1. Bilson JL Campana and Eamonn J Keogh. A compression-based distance measure for texture. *Statistical Analysis and Data Mining: The ASA Data Science Journal*, 3(6):381–398, 2010.
2. Peter D Grünwald and Abhijit Grunwald. *The minimum description length principle.* MIT press, 2007.
3. Ming Li, Paul Vitányi, et al. *An introduction to Kolmogorov complexity and its applications*, volume 3. Springer, 2008.
4. Hugo M Proença and Matthijs van Leeuwen. Interpretable multiclass classification by mdl-based rule lists. *arXiv preprint arXiv:1905.00328*, 2019.
5. Jorma Rissanen. Modeling by shortest data description. *Automatica*, 14(5):465–471, 1978.
6. Koen Smets and Jilles Vreeken. Slim: Directly mining descriptive patterns. In *Proceedings of the 2012 SIAM International Conference on Data Mining*, pages 236–247. SIAM, 2012.
7. Jilles Vreeken, Matthijs Van Leeuwen, and Arno Siebes. Krimp: mining itemsets that compress. *Data Mining and Knowledge Discovery*, 23(1):169–214, 2011.