1 Theoretical Framework

We define geometric pattern mining on bounded, discrete and two-dimensional raster-based 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). For 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.

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

Fig. 1: Example decomposition of A into instantiation I and patterns X, Y

According to MDL, the shortest (optimal) description of A reveals all structure of A in the most succinct way possible and we approximate this optimal description through the compression of the original data. This optimal description 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 defining A using as few building blocks as possible. We illustrate this in Figure 1. Given the matrix A we decompose it in patterns, denoted X and Y. The set of all these patterns is the **model** of a A, denoted H_A. In order to reconstruct A from this model, we alse need a mapping from the H_A back to A. This mapping represents in what MDL calls the **the data given the model** H_A. In this context we think of it 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 I_A in the example. The result is a notation that allows us to express matrix A as if decomposed into sets of local and global spatial information, which we will now describe in more detail.

1.1 Patterns and Instances

▷ *We define a* ***pattern*** *as a $M_X \times N_X$ submatrix X of the original matrix A. Elements of this submatrix may be $\cdot$, the empty element, which gives us the ability to 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 and that no rows and columns are empty.*

From this definition, the dimensions $M_X \times N_X$ give the smallest rectangle around X (the *bounding box)*. We 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.

Each pattern contains a special **pivot** element: *pivot*(X) is the first non-empty element of X. A pivot can be thought of as a fixed point in X which we can use to position its elements in relation to A. This translation, or **offset**, is a tuple $q = (i, j)$ 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 that the pivot element is at (i, j). We formalize this in the **instantiation operator** $\otimes$:

▷ *We define the* ***instance*** $X \otimes (i, j)$ *as the* $M \times N$ *matrix containing all elements of* X *such that* pivot(X) *is at index* (i, j) *and the distances between all elements are preserved. The resulting matrix contains no additional non-empty elements.*

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(X) *is at index* (i, j) *results in an instance 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.

▷ *Two pattern instances* $X \otimes q$ *and* $Y \otimes r$*, with* $q \neq r$ *are* ***non-overlapping*** *if* $|(X \otimes q) + (Y \otimes r)| = |X| + |Y|$*.*

From here on we will use the same letter in lower case to denote an arbitrary instance of a pattern, e.g. $x = X \otimes q$ when the exact value of q is unimportant.

We briefly introduced the instantiation I as a set of 'instructions' of where instances of each pattern should be positioned in order to obtain A. As Figure 1 suggests, this mapping has the shape of an $M \times N$ matrix.

▷ *Given the set of patterns* H*, the* ***instantiation (matrix)*** I *is an incomplete* $M \times N$ *matrix such that* $I_{i,j} \in H \cup \{\cdot\}$ *for all* (i, j)*. For all non-empty elements* $I_{i,j}$ *it holds that* $I_{i,j} \otimes (i, j)$ *is a non-overlapping instance of* $I_{i,j}$ *in* A*.*

The above definition creates the interesting proposition that the offset to each instance is unique. We provide a sketch proof in Appendix A.

1.2 The Problem and its Solution Space

Patterns can be constructed by joining smaller patterns in a bottom-up fashion. We can define the exact way two patterns should be joined by enumerating the distance of their respective pivots. To limit the possibilities to patterns relevant to A, instances can be used as an intermediate step. Since instances are simply patterns projected on an $M \times N$ matrix, we can reverse $\otimes$ by removing all completely empty rows and columns:

▷ *Let* $X \otimes q$ *be an instance of* X*, then by definition we say that* $\oslash(X \otimes q) = X$*.*

As Figure 2 demonstrates, we can use a simple element-wise matrix addition to sum two instances and use $\oslash$ to obtain a joined pattern. Here we start by instantiating X and Y with offsets $(1, 0)$ and $(1, 1)$ respectively. We add the resulting x and y to obtain $\oslash z$, the union of X and Y with relative offset $(1, 1) - (1, 0) = (0, 1)$.

Theorem 1. *Given two non-overlapping instances $x = X \otimes q$ and $y = Y \otimes r$, the sum of the matrices $x + y$ is another instance. We observe that pattern $Z = \oslash(x + y)$ such that $x + y = Z \otimes q$.*

The Sets $\boldsymbol{\mathcal{H}_A}$ and $\boldsymbol{\mathcal{I}_A}$ We define 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 $\mathcal{H}_A$ of all possible models for A, and $\mathcal{I}_A$, the set of all possible instantiations to these models. We will also take H^0_A to be the model with only singleton patterns. As singletons are just individual elements of A, we can simply say that $H^0_A = S$. The instantiation matrix corresponding to H^0_A is denoted I^0_A. Given that each element of this matrix must correspond to exactly one element of A in H^0_A, we see that each $I_{i,j} = a_{i,j}$ and so I^0_A is equal to A.

Using H^0_A and I^0_A as base cases we can now inductively define the set $\mathcal{I}_A$:

Base case $I^0_A \in \mathcal{I}_A$

By induction If I is in $\mathcal{I}_A$ then take any pair $I_{i,j}, I_{k,l} \in I$ such that $(i,j) \leq (k,l)$ in lexicographical order. Then the set I' is also in $\mathcal{I}_A$, providing I' equals I except:

$$I'_{i,j} := \oslash\big(I_{i,j} \otimes (i,j) + I_{k,l} \otimes (k,l)\big)$$
$$I'_{k,l} := \cdot$$

This shows we can add any two instances together, which are by definition always non-overlapping and thus valid in A, in any order and obtain an element of $\mathcal{I}_A$. Eventually this results in just one big instance that is equal to A. Note that when we take two elements $I_{i,j}, I_{k,l} \in I$ we force $(i,j) \leq (k,l)$, not only to eliminate different routes to the same instance matrix, but also such that the pivot of the new pattern coincides with $I_{i,j}$. We can then leave $I_{k,l}$ empty.

The construction of $\mathcal{I}_A$ also implicitly defines $\mathcal{H}_A$. While this may seem odd — defining models for instantiations instead of the other way around — note that there is no unambiguous way to find one instantiation for a given model. Instead we find the following definition by applying the inductive construction:

$$\mathcal{H}_A = \big\{\{\oslash(I) \mid I \in I\} \bigm| I \in \mathcal{I}_A\big\}. \tag{1}$$

So for any instantiation $I \in \mathcal{I}_A$ there is a corresponding set in $\mathcal{H}_A$ of all patterns that occur in I. This results in an interesting symbiosis between model and instantiation: increasing the complexity of one decreases that of the other. This construction gives a tightly connected lattice as shown in Figure 3.

$$x = X \otimes (1,0) = \begin{bmatrix} \cdot & \cdot \\ 1 & \cdot \\ \cdot & 1 \end{bmatrix},\ y = Y \otimes (1,1) = \begin{bmatrix} \cdot & \cdot \\ \cdot & 1 \\ \cdot & \cdot \end{bmatrix}, x + y = \begin{bmatrix} \cdot & \cdot \\ 1 & 1 \\ \cdot & 1 \end{bmatrix},\ Z = \oslash(x + y) = \begin{bmatrix} 1 & 1 \\ \cdot & 1 \end{bmatrix}$$

Fig. 2: Example of joining patterns X and Y to construct Z.

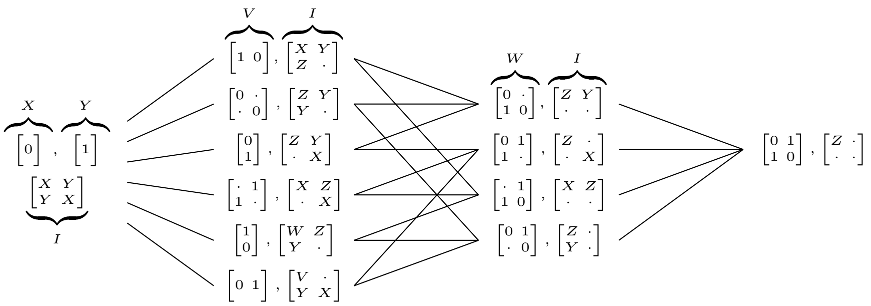

Fig. 3: The model space lattice for a 2×2 Boolean matrix. Here X and Y are the initial singleton patterns and I is the instance matrix. V and W are intermediate patterns and Z is the final, completely over fit, pattern.

1.3 Encoding Models and Instances

From all parametrized models in $\mathcal{H}_A$ we want to select (approximate) the model that describes A best. We use two-part MDL to quantify how well a given model and instantiation matrix fit A. Two-part MDL tells us to minimize the sum of $L_1(H_A) + L_2(A|H_A)$, two functions that give the length of the model and the length of 'the data given the model', respectively. In this context, the model is the set of patterns H_A and the data given the model is the accidental information needed to reconstruct the data from H_A — which in this case is the instantiation matrix I_A.

In order to give said length functions, we need to decide on a way to encode H and I first. This encoding is of great influence on the length functions and therefore on the ability to accurately quantify the fit of a given H and I to A. The decision for this encoding is obviously prone to bias, which is one of the few disadvantages of two-part MDL. Fortunately, practice shows that good results can be had once certain conditions are met: (1) all data is encoded (lossless) and (2) the encoding is as concise as possible (nothing but the data is encoded). Based on these conditions we give length functions for pattern sets and instance matrices, but we do not actually need to encode them.

The fictional encoder sequentially sends each symbol in the datastream (either pattern set or instance matrix) to the 'decoder' using a code word. Information theory tells us that we optimal length of a code word is given by $-\log(p)$, where p is the exact probability that the code word occurs in the output. We therefore need not compute the actual code words, just their probabilities. For this to work, both the encoder and hypothetical decoder must know either the exact probability distribution or agree upon an approximation beforehand. Such approximation is often called a **prior** and is used to fix 'prior knowledge' that does not have to be encoded explicitly.

To encode the instance 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 favorable property of being asymptotically equal to the optimal code for large sequences. Say we want to encode all elements $I_i \in I$, we define:

$$P_{plugin}(y_i = I_i \mid y^{i-1}) = \frac{|\{y \in y^{i-1} \mid y = I_i\}| + \epsilon}{\sum_{X \in H} |\{y \in y^{i-1} \mid y = X\}| + \epsilon} \tag{2}$$

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 $P_{plugin}(y_1 = I \mid y^0) = \frac{\epsilon}{\epsilon|H|}$. 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 I occurs.

▷ *Given a set of instances I, we define* $\text{usage}(X) = |\{I_i \in I \mid I_i = X\}|$.

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 1.3 to be able to encode items in arbitrary order. This produces the length function of the instance matrix I as follows:

$$\begin{aligned} L_{pp}(I \mid P_{plugin}) &= \sum_{i=1}^{|I|} -\log \frac{|\{y \in y^{i-1} \mid y = I_i\}| + \epsilon}{\sum_{X \in H} |\{y \in y^{i-1} \mid y = X\}| + \epsilon} \\ &= -\log \frac{\prod^{X_i \in H} \prod_{j=0}^{\text{usage}(X_i)} j + \epsilon}{\prod_{j=0}^{|I|-1} j + \epsilon|H|} \\ &= -\sum_{X_i \in h}^{|H|} \left[\log \frac{\mathit{\Gamma}(\text{usage}(X_i) + \epsilon)}{\mathit{\Gamma}(\epsilon)}\right] + \log \frac{\mathit{\Gamma}(|I| + \epsilon|H|)}{\mathit{\Gamma}(\epsilon|H|)} \end{aligned} \tag{3}$$

The length function for incomplete matrices. To losslessly encode A' we have to encode both H and I individually. Recall that both instances 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 behaves different for small sequences (patterns) than it does for large sequences (the instance matrix). Furthermore, we do not consider certain values such as the size of the instance 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.

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 $\log(MN)$. This is simply an uniform

Table 1: Length computation for the different classes of matrices. The total length is the sum of the listed terms.

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

distribution that 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 instance matrix, which encodes symbols to patterns, the prequential code is used as demonstrated before. Notice that L_N is the universal prior for integers[**?**] that can be used to encode integers on an arbitrary range.