Data Mining and Knowledge Discovery (2022) 36:108–145
https://doi.org/10.1007/s10618-021-00799-9

Mint: MDL-based approach for Mining INTeresting Numerical Pattern Sets

Tatiana Makhalova[1] · **Sergei O. Kuznetsov**[2] · **Amedeo Napoli**[1]

Received: 22 May 2020 / Accepted: 6 September 2021 / Published online: 4 October 2021

Abstract
Pattern mining is well established in data mining research, especially for mining binary datasets. Surprisingly, there is much less work about numerical pattern mining and this research area remains under-explored. In this paper we propose MINT, an efficient MDL-based algorithm for mining numerical datasets. The MDL principle is a robust and reliable framework widely used in pattern mining, and as well in subgroup discovery. In MINT we reuse MDL for discovering useful patterns and returning a set of non-redundant overlapping patterns with well-defined boundaries and covering meaningful groups of objects. MINT is not alone in the category of numerical pattern miners based on MDL. In the experiments presented in the paper we show that MINT outperforms competitors among which IPD, REALKRIMP, and SLIM.

Keywords Numerical Pattern Mining · Minimum Description Length principle · Plug-in codes · Numerical Data · Hyper-rectangles

1 Introduction

The objective of pattern mining is to discover a small set of interesting patterns that describe together a large portion of a dataset and can be easily interpreted and reused. Actually pattern mining encompasses a large variety of algorithms in knowledge dis-

Responsible editor: Ira Assent, Carlotta Domeniconi, Aristides Gionis, Eyke Hüllermeier

✉ Tatiana Makhalova
tatiana.makhalova@inria.fr

Sergei O. Kuznetsov
skuznetsov@hse.ru

Amedeo Napoli
amedeo.napoli@loria.fr

1 Université de Lorraine, CNRS, Inria, LORIA, F-54000 Nancy, France

2 National Research University Higher School of Economics, Moscow, Russia

covery and data mining aimed at analyzing datasets (Vreeken and Tatti 2014). Present approaches in pattern mining are aimed at discovering an interesting *pattern set* rather than a set of individually interesting patterns, where the quality of patterns is evaluated w.r.t. both the dataset and other patterns. One common theoretical basis for pattern set mining relies on the Minimum Description Length principle (MDL) (Grünwald 2007), which is applied to many types of patterns, e.g. itemsets (Vreeken et al. 2011), patterns of arbitrary shapes in 2-dimensional data (Faas and van Leeuwen 2020), sequences (Tatti and Vreeken 2012a), graphs (Bariatti et al. 2020), etc.

Contrasting the recent advances in pattern mining, algorithms for mining numerical data appear to be insufficiently explored. To date, one of the most common ways to mine numerical pattern sets relies on the application of itemset mining to binarized datasets. This is discussed below in more detail but before we would like to mention an alternative to numeric pattern mining which is "clustering".

In the last few decades, clustering algorithms have been extensively developed, and many different and efficient approaches have been proposed (Jain 2010; van Craenendonck et al. 2017; Jeantet et al. 2020). However, there is an important conceptual difference between pattern mining and clustering. In pattern mining the *description* comes first, while in clustering the primacy is given to the *object similarity*. In other words, numerical pattern mining is more interested in the description of a group of objects in terms of a set of attributes related to these objects, while clustering focuses more on the detection of these groups of objects based on their commonalities as measured by a similarity or a distance. The former entails some requirements for the ease of interpretation, i.e., the resulting patterns should describe a region in the "attribute space" that is easy to interpret. By contrast, in clustering, the focus is put on groups of objects or instances. The clusters can be constrained to have certain shapes, e.g., spheres in K- MEANS or DBSCAN, but still the similarity of objects remains the most important characteristic of clusters. For example, clustering techniques such as agglomerative single-linkage clustering in a multidimensional space may return clusters of very complex shapes. Usually no attention is paid to these shapes while this is one of the most important preoccupations in numerical pattern mining.

Accordingly, in this paper, we propose an MDL-based approach to numerical pattern set mining called MINT for "Mining INTeresting Numerical Pattern Sets". MINT computes numerical patterns as m-dimensional hyper-rectangles which are products of m intervals, where the intervals are related to the attributes and their values. MINT does not require any frequency threshold. For each pattern the frequency threshold is adapted according to the MDL principle and depends on the size of the pattern, pattern neighbors, and the size of the pattern set. The main benefits of the present approach are that (i) MINT does not need to explore the pattern space in advance as candidates for optimal patterns are computed on the fly, (ii) the total number of explored patterns is at most cubic (and it is at most quadratic at each iteration) in the number of objects with distinct descriptions considered as vectors of attribute values, (iii) MINT is based on MDL and outputs a small set of non-redundant informative patterns, (iv) a series of experiments shows that MINT is efficient and outputs sets of patterns of very high quality. The discovered patterns are diverse, non-redundant, cover almost entirely the whole dataset by a reasonable (i.e., relatively small) number of patterns. Furthermore they describe meaningful groups of objects with quite precise boundaries.

Actually, there are two versions of MINT, the first one is based on a covering strategy while the second one, called GMINT for GREEDY MINT, is based on a greedy approach when computing the candidate patterns. The results returned by both versions of MINT algorithm are quite close but the GMINT version is much faster. Moreover, MINT algorithm is able to mine numerical patterns both for small and on large datasets, and it is most of the time more efficient, outputting more concise and better fitting patterns, as compared to its competitors IPD (an MDL-based method for discretization), REALKRIMP, and SLIM. In particular, the encoding scheme on which relies MINT is based on prequential plug-in codes, which have better theoretical properties than the codes used in IPD, REALKRIMP, and SLIM.

The paper has the following structure. In Sect. 2 we discuss the state-of-the-art algorithms in pattern mining for numerical data. Section 3 introduces the main notions used in the paper while in Sect. 4 we describe the basis of the proposed method. Next Sect. 5 relates the experiments carried out for illustrating the good behavior and the strengths of MINT. Finally, Sect. 7 concludes the paper with a discussion about the potential of MINT and some directions for future work.

2 Related work

The problem of pattern mining has been extensively studied for binary data –itemset mining– but remains much less explored for numerical data. Hence a common way to mine patterns in numerical data relies on a binarization of data and then application of itemset mining algorithms. Meanwhile, a number of approaches was designed for mining numerical datasets possibly involving binarization and taking into account the type of the data at hand. In this section we firstly discuss different numerical data preprocessing approaches allowing the use of itemset mining and then we discuss state-of-the art approaches in numerical pattern mining.

2.1 Numerical data preprocessing

Data preprocessing is the cornerstone for discovering patterns of good quality and relies on *discretization* or *binarization* tasks.

Discretization. Discretization relies on partitioning the range of attribute values into intervals and then mapping the intervals into integers for preserving the order of the intervals. The existing discretization techniques can be categorized into *univariate* and *multivariate* techniques.

Univariate discretization includes all the methods where attributes are considered independently. An attribute range may be split into intervals of equal width or equal height w.r.t. frequency. Another way to split an attribute range is by using K- MEANS algorithm (Dash et al. 2011), where some criteria are used for assessing the quality of clustering and choosing an optimal K. A more flexible approach consists in splitting based on the MDL principle (Kontkanen and Myllymäki 2007; Rissanen et al. 1992) which is discussed below.

However, considering each attribute range independently does not preserve the interaction between attributes and, as a consequence, may make some patterns not recognizable. Multivariate discretization techniques were proposed to tackle this issue. In (Mehta et al. 2005; Kang et al. 2006), multivariate discretization is based on principal component analysis and independent component analysis, respectively. However, both techniques do not guarantee taking into account possible complex interactions between attributes and require some assumptions on either distribution or correlation (Nguyen et al. 2014). Accordingly, Nguyen et al. (2014) address these problems proposing an MDL-based algorithm, called IPD, for multivariate discretization. The algorithm works in unsupervised settings contrasting a related approach in (Fayyad and Irani 1993; Boullé 2006; Bondu et al. 2010). Indeed, MDL is used in a large number of approaches and is detailed below in § 2.2.

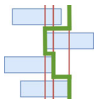
Binarization. Discretization is not the only step to accomplish before applying an itemset mining algorithm. Another important operation is binarization, i.e., the transformation of discrete values into binary attributes. Binarization should be carefully taken into account as it may affect the results of itemset mining and induce loss of information. Moreover, binarization is associated with the risk of introducing artifacts and then obtaining meaningless results.

A simple and popular binarization is based on "one-hot encoding", where each discrete value is associated with one binary attribute. The number of attributes may become very large which can lead to an excessive computational load. Moreover, one-hot encoding does not necessarily preserve the order of discrete values.

By contrast, interordinal scaling preserves the order of values by introducing for each discrete value v two binary attributes "$x \geq v$" and "$x \leq v$". However, in (Kaytoue et al. 2011) it was shown that, with a low-frequency threshold, mining itemsets in interordinally scaled data becomes much more expensive than mining hyper-rectangles in numerical data. Hyper-rectangles here are studied in the framework of interval pattern structures.

An alternative approach to interordinal scaling (Srikant and Agrawal 1996) consists in computing more general itemsets based on considered discrete values. The authors introduce the notion of partial completeness w.r.t. a given frequency threshold. This notion allows one to formalize the information loss caused by partitioning as well as to choose an appropriate number of intervals w.r.t. chosen parameters. As interordinal scaling, this approach suffers from pattern explosion. In addition, this method requires to set some parameters, e.g., frequency threshold and completeness level, whose optimal value is unknown.

Despite its limitations, one-hot encoding remains a good option provided that suitable constraints can be defined for avoiding attribute explosion and allowing tractable computation.



Numerical attribute set assessment based on ranks. One of the main drawbacks of the approaches mentioned above, which consider discretization and binarization as mandatory preprocessing steps, is that the quality of the output depends on the quality

of the discretization. In mining numerical patterns, when the boundaries (given in red) of patterns are not well aligned with vertical axes, as shown in the figure on the right, uniform boundaries will produce imprecise descriptions, while using exact boundaries may greatly complicate pattern mining.

An alternative approach that "simulates" multi-threshold discretization is proposed in a seminal paper (Calders et al. 2006). It consists in (i) considering the ranks of attribute values instead of actual real values, and (ii) evaluating the sets of numerical attributes using rank-based measures. The authors propose several support measures based on ranks. In such a way, the problem of dealing with concrete attribute values is circumvented by considering the coherence of the ranked attribute values. Moreover, in (Tatti 2013), two scores based on ranked attribute values have been proposed to evaluate a set of numerical attributes. The scores are used to find the best combinations of attributes w.r.t. rank-based supports.

In all the methods mentioned in this subsection, patterns are understood as combinations of the attribute ranges as a whole. These methods do not provide descriptions related to some particular parts of the range if needed, while this is the main focus of this paper as explained later.

2.2 MDL-based approaches to pattern mining

The MDL approach to pattern set mining is based on the slogan: "the best set of patterns is the set that compresses the database best" (Grünwald 2007). There is a significant amount of papers about the use of the MDL principle in pattern mining as this is very well presented in the report of Galbrun (Galbrun 2020). MDL has been used in many different contexts but hereafter we focus on pattern mining. One of the most famous MDL-based itemset miners is KRIMP, introduced in (Siebes et al. 2006) (not under this name), and the last version was presented in (Vreeken et al. 2011). KRIMP relies on two steps that consist in (i) generating a set of frequent patterns, and (ii) selecting those minimizing the total description length. While KRIMP is an efficient and well-designed itemset miner, it requires that all frequent itemsets should be generated. Moreover, increasing the frequency threshold may lead to a worse compression, so SLIM (Smets and Vreeken 2012) was proposed to tackle this issue. In contrast to KRIMP, SLIM does not require that all itemsets should be generated in advance, since the candidates for optimal itemsets are gradually discovered. Nevertheless, the encoding scheme used in KRIMP and SLIM shows a range of limitations that are discussed in more detail in Sect. 3.2. In continuation, the DIFFNORM algorithm (Budhathoki and Vreeken 2015) is an extension of SLIM that is based on a better encoding scheme and can be applied to a collection of datasets for finding the difference in datasets in terms of itemsets. Another MDL algorithm related to the KRIMP family was proposed in (Akoglu et al. 2012) for fixing scalability issues. This algorithm deals with categorical data and is less sensitive to combinatorial explosion.

All the aforementioned MDL-based algorithms represent a "model", i.e. a set of patterns, as a two-column table, where the left-hand column contains the pattern descriptions and the right-hand column contains the associated code words. Another way to store patterns is proposed in the PACK algorithm (Tatti and Vreeken 2008),

where the model is encoded as a decision tree, so that a node corresponds to an attribute. A non-leaf node has two siblings reflecting the presence or absence of this attribute in an itemset. The itemset, in turn, is a path from the root to a leaf node. One main difference between the PACK approach and the algorithms of the KRIMP family is that 0's and 1's are symmetrically considered in PACK.

The STIJL algorithm (Tatti and Vreeken 2012b) is a tree-based MDL approach taking into account both 0's and 1's and storing itemsets in a tree. However, contrasting PACK, STIJL relies on "tiles", i.e., rectangles in a dataset. The tree in STIJL is a hierarchy of nested tiles, where parent-child relations are inclusion relations between tiles. A child tile is created whenever its density –the relative number of 1's– differs a lot from the parent one. An extension of tile discovery is proposed in (Faas and van Leeuwen 2020) where "geometric pattern mining" with the VOUW algorithm is introduced. This algorithm may consider arbitrarily shaped patterns in raster-based data, i.e., data tables with a fixed order of rows and columns, and it is able to identify descriptive pattern sets even in noisy data. Finally, the discovery of tiles is also closely related to Boolean Matrix Factorization (BMF). In a nutshell, the objective of BMF is to find an approximation of a binary matrix C by a Boolean product of two low-rank matrices A and B. The columns of A and the rows of B describe the factors, which correspond to tiles. The MDL principle can also be applied to the BMF problem (Miettinen and Vreeken 2014; Makhalova and Trnecka 2021).

All the MDL-based algorithms which are surveyed above are applicable to binary or categorical data. Now we focus on a few algorithms which are dealing with pattern mining in numerical data. First of all the REALKRIMP algorithm (Witteveen et al. 2014) is an extension of KRIMP to real-valued data, where patterns are axis-aligned hyper-rectangles. Even if the algorithm does not require any preprocessing, it actually needs a discretization of the data. Moreover, there is also a "feature selection" step where unimportant hyper-rectangle dimensions are removed. REALKRIMP is tailored to mine high-density patterns, and to minimize the combinatorial explosion, it constructs each hyper-rectangle using a pair of neighboring rows sampled from the original dataset. Then, without prior knowledge about the data, the choice of the size of sampling is difficult as a too small sample size may output very general patterns, while a too large sample size may increase the execution time. The problem of an inappropriate sample size may be partially solved by setting a large "perseverance", i.e., how many close rows should be checked to improve compression when enlarging the hyper-rectangle, and "thoroughness", i.e., how many consecutive compressible patterns are tolerated. As it can be understood, finding optimal parameters in REALKRIMP constitutes an important problem in the pattern mining process. Moreover, the hyper-rectangles in REALKRIMP are evaluated independently, meaning that the algorithm searches for a set of optimal patterns instead of an optimal pattern set. The subsequent pattern redundancy may be mitigated by sampling data and computing the hyper-rectangles in different parts of the attribute space. Thereby, REALKRIMP relies on many heuristics and has no means to jointly evaluate the set of generated hyper-rectangles. In addition, heuristics imply some prior knowledge about the data which is not always available in practice. All these aspects should be taken into account.

Another approach to mine informative hyper-rectangles in numerical data was proposed in (Makhalova et al. 2019). The approach can be summarized in 3 steps: (i)

greedily computing dense hyper-rectangles by merging the closest neighbors and ranking them by prioritizing dense regions, (ii) greedily optimizing an MDL-like objective to select the intervals –sides of hyper-intervals– for each attribute independently, (iii) constructing the patterns using the selected intervals and maximizing the number of instances described by the intervals by applying a closure operator (a closed set is maximal for given support). This approach tends to optimize entropy, which is proportional to the length of data encoded by the intervals and does not take into account the complexity of the global model, i.e., the set of patterns. This simplification is based on the observation that each newly added interval replaces at least one existing interval, and thus, the complexity of the model does not increase. Moreover, the compression process is lossy as the data values can be reconstructed only up to some selected intervals. Finally, the approach allows for feature selection but does not address explicitly the problem of overlapping patterns.

Based on this first experience, below we propose the MINT algorithm, which is based on MDL and aimed at mining patterns in numerical data. We restrict the patterns to be hyper-rectangles as they are the most interpretable types of multidimensional patterns.

As REALKRIMP and IPD, MINT deals with discretized data. Both REALKRIMP and MINT allow for mining overlapping patterns, however the problem of feature selection is not addressed in MINT. MINT is less dependent on heuristics than REALKRIMP and discovers an approximation of an MDL-optimal pattern set rather than an approximation of single optimal patterns.

IPD works in a setting similar to MINT, however, the methods differ in several aspects: (i) IPD searches for globally optimal boundaries (in experiments we show that these boundaries are not quite precise to describe "ground-truth" hyper-rectangles), (ii) IPD returns a grid where adjacent hyper-rectangles may belong to one "ground-truth" hyper-rectangle. Overall, despite the fact that both algorithms deal with numerical data and are aimed at finding meaningful subspaces using the MDL principle, MINT and IPD solve very distinct tasks. At each step IPD decides whether a particular boundary is useless or not, while MINT decides whether a particular hyper-rectangle is different enough from its neighbors and other hyper-rectangles to be considered as a separate pattern. As a consequence, both the total description length and the principles of its minimization are very different in both approaches.

In MINT we introduce the total description length for a set of hyper-rectangles. The proposed encoding adapts some of the best practices of the aforementioned approaches: (i) prequential plug-in codes for patterns (Budhathoki and Vreeken 2015; Proença and van Leeuwen 2020), (ii) grid-based encoding of the boundaries and reconstruction cost to refine the positions of data points within the hyper-rectangles (Nguyen et al. 2014). In contrast to (Tatti and Vreeken 2008), the patterns are not arranged into a hierarchy and their boundaries are encoded independently from the other patterns in the set. However, patterns still may overlap and even be included in other patterns. As SLIM, MINT discovers gradually the candidates for optimal patterns and estimates the length gain to pick the best candidate.

3 Basics

3.1 Formalization of data and patterns

Let D^* be a numerical dataset that consists of a set of objects $G = \{g_1, \ldots, g_n\}$ and a set of attributes $M = \{m_1, \ldots, m_k\}$. The number of objects and attributes is equal to n and k, respectively. Each attribute $m_i \in M$ is numerical and its range of values is denoted by $range(m_i)$. Each object $g \in G$ is *described* by a tuple of attribute values $\delta(g) = \langle v_i \rangle_{i \in \{1,\ldots,k\}}$.

As patterns we use axis-aligned hyper-rectangles, or "boxes". In multidimensional data, an axis-aligned hyper-rectangle has one of the simplest descriptions –a tuple of intervals– and thus can be easily analyzed by humans. The hyper-rectangle describing a set of objects B is given by

$$h = \langle [\min\{v_i \mid v_i \in \delta(g), g \in B\}, \max\{v_i \mid v_i \in \delta(g), g \in B\}] \rangle_{i \in \{1,\ldots,k\}} .$$

We call the i-th interval of a hyper-rectangle the i-th *side* of the hyper-rectangle. The support of a hyper-rectangle h is the number of objects whose descriptions comply with h, i.e., $sup(h) = |\{g \in G \mid \delta(g) \in h\}|$.

Often, instead of continuous numerical data, one deals with discretized data, where the continuous range of values $range(m_i)$ is replaced by a set of integers, which are the indices of the intervals. Formally speaking, a range $range(m_i)$ is associated with a partition based on a set of intervals $\mathcal{B}_i = \{B_i^j = [c_{j-1}, c_j) \mid j = 1, \ldots, l\}$, where c_0 and c_l are the minimum and maximum values, respectively, of $range(m_i)$. Thus, each $v \in [c_{j-1}, c_j)$ is replaced by j.

The endpoints of the intervals can be chosen according to one of the methods considered above, e.g., equal-width, equal-height intervals or using the MDL principle, and the number of the intervals may vary from one attribute range to another attribute range. The endpoints make a *discretization grid*. The number of the grid dimensions is equal to the number of attributes.

A chosen discretization splits the space $\prod_{i=1}^{|M|} range(m_i)$ into a finite number of elementary hyper-rectangles $h_e \in \{\prod_{i=1}^{|M|} B_i^j \mid B_i^j \in \mathcal{B}_i\}$, i.e., each side of an elementary hyper-rectangle is composed of one discretization interval B_i^j. Non-elementary hyper-rectangles are composed of consecutive elementary hyper-rectangles.

For a hyper-rectangle $h = \left\langle [c_i^{(l)}, c_i^{(u)}) \right\rangle_{i \in \{1,\ldots,|M|\}}$, where $c_i^{(l)}$ and $c_i^{(u)}$ are endpoints of intervals from $\mathcal{B}_i$, in the discretized attribute space we define the *size* of the ith side as the number of elementary hyper-rectangles included into this side, i.e., $size(h, i) = |\{B_i^j \mid B_i^j \subseteq [c_i^{(l)}, c_i^{(u)})\}|$. Further, we use h to denote a hyper-rectangle (pattern), $\mathcal{H}$ to denote a set of hyper-rectangles (patterns), and D to denote the dataset D^* discretized w.r.t. the chosen discretization grid.

Example 1 Let us consider a dataset given in Fig. 1 (left). It consists of 13 objects described by attributes m_1 and m_2. All the descriptions are distinct (unique). Each attribute range is split into 8 intervals of width 1. The discretized dataset is given in Fig. 1 (right). It has 7 unique rows. The non-empty elementary hyper-rectangles

	m_1	m_2	m_1	m_2	
g_1	0.30	0.15	0	0	h_1
g_2	0.20	0.35	0	0	
g_3	0.05	3.90	0	4	h_2
g_4	4.40	0.00	4	0	h_3
g_5	4.30	3.70	4	4	h_4
g_6	4.25	3.55	4	4	
g_7	4.10	3.90	4	4	
g_8	4.25	6.60	4	7	h_5
g_9	6.10	4.15	6	4	h_6
g_{10}	6.45	3.75	6	4	
g_{11}	5.70	6.50	6	7	h_7
g_{12}	5.90	7.40	6	7	
g_{13}	6.10	7.35	6	7	

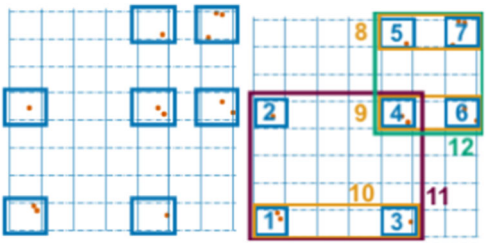


Fig. 1 Dataset over attributes $\{m_1, m_2\}$ and its discretized version (left), representation of the dataset in the plane and its partition into 7×8 equal-width intervals (right). The discretization grid is given by dotted lines, the corresponding non-empty elementary hyper-rectangles are given by dashed lines. The axis labels show the indices of elementary hyper-rectangles

correspond to non-empty rectangles induced by the 7×8 discretization grid. The number of hyper-rectangles is equal to the number of distinct rows in the discretized dataset (given in the middle).

3.2 Information theory and MDL

MDL (Grünwald 2007) is a general principle that is widely used for model selection and works under the slogan "the best model compresses data the best". This principle relies on the following: given a sequence that should be sent from a transmitter to a receiver, the transmitter, instead of encoding each symbol uniformly, replaces repetitive sub-sequences with code words. Thus, instead of a symbol-wise encoded sequence, the transmitter sends a sequence of code words and a dictionary. The dictionary contains all used code words and the sub-sequences encoded by them. Using the dictionary, the receiver is able to reconstruct the original sequence. The MDL principle is applied to decide which sub-sequences should be replaced by the code words and which code words should be chosen for these sub-sequences. The code words are associated in such a way that the most frequent sub-sequences have the shortest code words. In our case, the sequence that should be transmitted correspond to a numerical dataset discretized into small equal-width intervals, and as sub-sequences we chose patterns (hyper-rectangles).

Formally speaking, given a dataset D the goal is to select a subset of patterns $\mathcal{H}$ that minimizes the description length $L(D, \mathcal{H})$. In the crude version of MDL (Grünwald 2007) the description length is given by $L(D, \mathcal{H}) = L(\mathcal{H}) + L(D|\mathcal{H})$, where $L(\mathcal{H})$ is the description length of the model (set of patterns) $\mathcal{H}$, in bits, and $L(D|\mathcal{H})$ is the description length of the dataset D encoded with this set of patterns, in bits.

The length $L(\mathcal{H})$ characterizes the complexity of the set of patterns and penalizes high-cardinality pattern sets, while the length of data $L(D|\mathcal{H})$ characterizes the conformity of patterns w.r.t. the data. $L(D|\mathcal{H})$ increases when the patterns are too general

and do not conform well with the data. Thus, taking into account both $L(\mathcal{H})$ and $L(D|\mathcal{H})$ allows to achieve an optimal balance between the pattern set complexity and its conformity with the data.

Roughly speaking, the minimization of the total length consists in (i) choosing patterns that are specific for a given dataset, and (ii) assigning to these patterns the code words allowing for a shorter total length $L(D, \mathcal{H})$.

In MDL, our concern is the length of the code words rather than the codes themselves. That is why we use a real-valued length instead of an integer-valued length.

Intuitively, the length of code words is optimal when shorter code words are assigned to more frequently used patterns. From the information theory, given a probability distribution over $\mathcal{H}$, the length of the Shannon prefix code for $h \in \mathcal{H}$ is given by $l(h) = -\log P(h)$ and is optimal. Then we obtain the following probability model: given the usage $usg(h)$ of $h \in \mathcal{H}$ in the encoding. The probability distribution ensuring an optimal pattern code length for the chosen encoding scheme is $P(h) = \frac{usg(h)}{\sum_{h_i \in \mathcal{H}} usg(h_i)}$, where usage $usg(h)$ of a pattern h is the number of times a pattern h is used to cover objects G in a dataset D. However, this model is based on the assumption that the total number of encoded instances (the length of the transmitting sequence) is known. Moreover, in order to encode/decode the message, the transmitter should know the usages $usg(h)$ of all patterns $h \in \mathcal{H}$ and the receiver should know the corresponding probability distribution, which is not usually the case.

Prequential plug-in codes (Grünwald 2007) do not have this kind of limitation. These codes refer to "online" codes since they can be used to encode sequences of arbitrary lengths and they do not require to know in advance the usage of each code word. The codes are based on only previously encoded instances. Moreover, they are asymptotically optimal even without any prior knowledge on the probabilities. The prequential plug-in codes are widely used in recent MDL-based models (Faas and van Leeuwen 2020; Proença and van Leeuwen 2020; Budhathoki and Vreeken 2015).

More formally, the idea of the prequential codes is to assess the probability of observing the n-th element h^n of the sequence based on the previous elements $h^1, \ldots, h^{n-1}$. Thus prequential codes allow for a predictive-sequential interpretation for sequences of arbitrary lengthsi.

Let H^n be the sequence $h^1, \ldots, h^{n-1}, h^n$. The probability of the n-th pattern $h^n \in \mathcal{H}$ in the pattern sequence H^n is given by

$$P_{plug\text{-}in}(h^n) = \frac{\prod_{h \in \mathcal{H}} [\Gamma(usg(h) + \varepsilon)/\Gamma(\varepsilon)]}{\Gamma(usg(\mathcal{H}) + \varepsilon|\mathcal{H}|)/\Gamma(\varepsilon|\mathcal{H}|)}, \tag{1}$$

where $usg(h)$ is the number of occurrences of pattern h in the sequence H^n, and $usg(\mathcal{H}) = \sum_{h \in \mathcal{H}} usg(h)$ is the length of the sequence, i.e., the total number of occurrences of patterns from $\mathcal{H}$. ε is a pseudocount, i.e., the initial usage of patterns, and $\Gamma(x) = \int_0^1 (-\log(t))^{x-1}\, dt$ is the gamma function.

Then the length of the code word associated with h^n is given as follows:

$$\begin{aligned} l(h^n) &= -\log P_{plug\text{-}in}(h^n) = \\ &= \log \Gamma(usg(\mathcal{H}) + \varepsilon|\mathcal{H}|) \end{aligned}$$

$$- \log \Gamma(\varepsilon|\mathcal{H}|) - \sum_{h \in \mathcal{H}} \left[\log \Gamma(usg(h) + \varepsilon) - \log \Gamma(\varepsilon)\right]. \quad (2)$$

As was mentioned above, we are interested in the *length* of the code words rather than in the code words themselves. That is why we use real-valued length instead of integer-valued length for the number of bits needed to store the real code words. We give the technical details of the derivation of Equation 1 in Appendix A.

To encode integers, when it is needed, we use the *standard universal code for the integers* (Rissanen 1983) given by $L_{\mathbb{N}}(n) = \log n + \log\log n + \log\log\log n + \ldots + \log c_0$, where the summation stops at the first negative term, and $c_0 \approx 2.87$ (Grünwald 2007). In this paper we write log for $\log_2$ and put $0 \log 0 = 0$.

4 Mint

We propose an approach to pattern mining in multidimensional numerical discretized data. The main assumption on which we rely is that all the attributes are equally important, i.e., patterns are computed in the whole attribute space. To apply this method we consider a discretized attribute space, i.e., each attribute range is split into equal-width intervals, as it was done in (Witteveen et al. 2014; Nguyen et al. 2014). The choice of equal-width intervals is due to the fact that the cost, in bits, of the reconstruction of a real value here is constant for all intervals. Each object therefore is included into an $|M|$-dimensional *elementary hyper-rectangle*. Starting from the elementary hyper-rectangles (each side is composed of one interval), we greedily generalize the currently best patterns and select those that provide the maximal reduction of the total description length. At each step we reuse some of the previously discovered candidates as well as other candidates computed on the fly using the last added pattern.

4.1 The model encoding

Firstly, we define the total description length of the set of hyper-rectangles and the data encoded by them. The total description length is given by $L(D, \mathcal{H}) = L(\mathcal{H}) + L(D|\mathcal{H})$, where $L(\mathcal{H})$ is the description length, in bits, of the set of hyper-rectangles $\mathcal{H}$, and $L(D|\mathcal{H})$ is the description length, in bits, of the discretized dataset encoded by this set of hyper-rectangles. The initial set of the hyper-rectangles is composed exclusively of elementary hyper-rectangles.

To encode the set of hyper-rectangles $\mathcal{H}$, we need to encode the discretization grid and the positions of the hyper-rectangles in this grid. Thus, the total length of the pattern set is given by

$$L(\mathcal{H}) = \underbrace{L_{\mathbb{N}}(|M|) + \sum_{i=1}^{|M|} L_{\mathbb{N}}(|\mathcal{B}_i|)}_{length\ of\ the\ grid} + \underbrace{L_{\mathbb{N}}(|\mathcal{H}|) + |\mathcal{H}| \left(\sum_{i=1}^{|M|} \log\left(|\mathcal{B}_i|(|\mathcal{B}_i|+1)/2\right) \right)}_{length\ of\ the\ pattern\ set}.$$

To encode the grid we need to encode the number of dimensions (attributes) $|M|$ and the number of intervals $|\mathcal{B}_i|$ within each dimension i. This grid is fixed and is not changed throughout the pattern mining process. To encode the pattern set $\mathcal{H}$, given the grid, we need to encode the number of patterns $|\mathcal{H}|$ and the positions of their boundaries within each dimension. Since there exist $\binom{|\mathcal{B}_i|}{2} + |\mathcal{B}_i|$ possible positions of the boundaries within the i-th dimension, namely $\binom{|\mathcal{B}_i|}{2}$ combinations where the boundaries are different, and $|\mathcal{B}_i|$ cases where the lower and upper boundaries belong to the same interval, meaning only one interval from the grid is involved. These positions are encoded uniformly.

The size of the intervals is taken into account in the reconstruction cost $L(D \ominus D(\mathcal{H})|\mathcal{H})$ (see below). The latter gives the cost of $\log(|\mathcal{B}_i|(|\mathcal{B}_i|+1))$ bits for encoding the i-th side of a pattern within the chosen grid.

The code length of a dataset encoded with the set of patterns is given by

$$L(D|\mathcal{H}) = \underbrace{L_{\mathbb{N}}(|G|)}_{\substack{\text{cost of encoding the} \\ \text{number of instances}}} + \underbrace{L(D(\mathcal{H})|\mathcal{H})}_{\substack{\text{cost of encoding} \\ D \text{ with } \mathcal{H}}} + \underbrace{L(D \ominus D(\mathcal{H})|\mathcal{H})}_{\text{reconstruction cost}}$$

where the first component encodes the number of objects, the second one corresponds to the length of data encoded with hyper-rectangles, and the third one corresponds to the cost of the reconstruction of the object description $\delta(g) = \langle v_i \rangle_{i \in \{1,\ldots,|M|\}}$ up to elementary intervals. Let us consider the last two components in more detail.

The cost of the reconstruction of the true real values is constant for all values due to the equal-width discretization and it is not changed during pattern mining, thus is not taken into account in $L(D, \mathcal{H})$. The dataset is encoded by exploring all objects in a given order and assigning to each object a code word of the pattern covering this object. According to the MDL principle, each data fragment should be covered (encoded) only once, otherwise, the encoding is redundant. However, some patterns may overlap, i.e., include the same object. A *cover strategy* then defines which data fragment is an occurrence of which pattern. We discuss the cover strategy in detail in the next section. Here, the usage is defined as follows: $usg(h) = |cover(h, G)|$.

From Equation 2, the length of data encoded with the plug-in codes is

$$\begin{aligned} L(D(\mathcal{H})|\mathcal{H}) &= \log \frac{\Gamma(usg(\mathcal{H}) + \varepsilon|\mathcal{H}|)/\Gamma(\varepsilon|\mathcal{H}|)}{\prod_{h\in\mathcal{H}} \Gamma(usg(h) + \varepsilon)/\Gamma(\varepsilon)} = \\ &= \log \Gamma(usg(\mathcal{H}) + \varepsilon|\mathcal{H}|) - \log \Gamma(\varepsilon|\mathcal{H}|) - \sum_{h\in\mathcal{H}} \big[\log \Gamma(usg(h) + \varepsilon) \\ &\quad - \log \Gamma(\varepsilon)\big], \end{aligned}$$

where $usg(\mathcal{H}) = \sum_{h\in\mathcal{H}} usg(h)$.

Once each object has been associated with a particular pattern, its original description within the pattern up to elementary intervals is encoded in $L(D \ominus D(\mathcal{H})|\mathcal{H})$. We use $D \ominus D(\mathcal{H})$ to denote the difference ("distortion") between the initially discretized dataset D and the same dataset encoded with $\mathcal{H}$.

To reconstruct the dataset up to the elementary equal-width intervals we encode the positions of each object within the corresponding pattern, this cost is

$$L(D \ominus D(\mathcal{H})|\mathcal{H}) = \sum_{h\in\mathcal{H}} usg(h)\log(size(h)) = \sum_{h\in\mathcal{H}} usg(h)\Big(\sum_{i=1}^{|M|}\log(size(h,i))\Big),$$

where $size(h,i)$ is the number of elementary intervals that compose the side i of the pattern h.

Example 2 Let us consider an encoding of the data by patterns according to the model introduced above for the case of the running example. We take the set of two hyper-rectangles $\mathcal{H} = \{h_{11}, h_{12}\}$, which are given in Fig. 1. Let the cover of h_{11} be $cover(h_{11}, G) = \{g_1, g_2, g_3, g_4\}$ and cover of h_{12} be $cover(h_{12}, G) = \{g_5, \ldots, g_{13}\}$. Then, the encoding of the pattern set is given by $L(\mathcal{H}) = L_{\mathbb{N}}(2) + (L_{\mathbb{N}}(7) + L_{\mathbb{N}}(8)) + L_{\mathbb{N}}(2) + 2\cdot(\log 28 \cdot \log 36)$. Here, we need $\log 28 \cdot \log 36$ bits to encode each pattern, i.e., $\log n$ bits to encode uniformly n possible positions of the boundaries for each side of the hyper-rectangle.

The length of data encoded by $\mathcal{H}$ is given by $L(D(\mathcal{H})) = L_{\mathbb{N}}(13) + \log\Gamma(13 + \varepsilon\cdot 2) - \log\Gamma(\varepsilon\cdot 2) - [\log\Gamma(8+\varepsilon) - \log\Gamma(\varepsilon) + \log\Gamma(4+\varepsilon) - \log\Gamma(\varepsilon)]$. The reconstruction error is equal to $L(D \ominus D(\mathcal{H})) = 9\cdot(\log(3) + \log(4)) + 4\cdot(\log(5) + \log(5))$, i.e., we need to encode the positions of the data points within the corresponding hyper-rectangles up to the elementary hyper-rectangles.

As we can see from the example above, the patterns can overlap. In such a case, one relies on a cover strategy to decide which pattern to use to encode each object. In the next section we introduce the algorithm that defines this strategy and allows for computing patterns minimizing the total description length.

4.2 The MINT algorithms

The objective of the MINT algorithm is to compute in a numerical dataset a pattern set which is the best w.r.t. the MDL principle.

4.2.1 Computing minimal pattern set

Let M be a set of continuous attributes, G be a set of objects having a description based on attributes M, $\mathcal{P}$ be a set of all possible $|M|$-dimensional hyper-rectangles defined in the space $\prod_{m\in M} range(m)$, and $cover$ be a cover strategy. One main problem is to find the smallest set of hyper-rectangles $\mathcal{H} \subseteq \mathcal{P}$ such that the total compressed length $L(D, \mathcal{H})$ is minimal.

The pattern search space in numerical data, where patterns are hyper-rectangles, is infinite. Even considering a restricted space, where all possible boundaries of the hyper-rectangles are limited to the coordinates of the objects from G, the search space is still exponentially large. The introduced total length $L(D, \mathcal{H})$ does not exhibit (anti)monotonicity property over the pattern set and thus does not allow us to exploit

some efficient approaches to its minimization. Hence, to minimize $L(D, \mathcal{H})$, we resort to heuristics.

4.2.2 Cover strategy

As mentioned above, some hyper-rectangles (patterns) may overlap. To ensure the minimality of encoding, i.e., that each object (data point) is exclusively covered by one pattern, we introduce a cover strategy. We call *log-volume* of pattern h the number of elementary hyper-rectangles in logarithmic scale that h contains, namely $lvol(h) = \sum_{i=1}^{|M|} \log size(h, i)$. The support of h is the number of objects from G that h contains, i.e., $sup(h, G) = \{g \in G \mid g \in h\}$. For the sake of simplicity, we write $sup(h)$ instead of $sup(h, G)$ when the support is computed on the whole set of objects. We say that a point $(v_1, \ldots v_{|M|})$ is *lexicographically smaller* than a point $(w_1, \ldots w_{|M|})$ if for the first position i where $v_i \neq w_i$ the following inequality holds: $v_i < w_i$ (for all the preceding elements $v_j = w_j$, $j = 1, \ldots, i-1$). Then h_1 is lexicographically smaller than h_2 if the lexicographically smallest point of h_1 is lexicographically smaller than the lexicographically smallest point of h_2.

The *standard cover order* of hyper-rectangles is given as follows:

$$lvol \uparrow, \quad sup \downarrow, \quad \text{lexicographically} \uparrow,$$

where $\uparrow$ / $\downarrow$ denote ascending / descending orders, respectively. Thus, the patterns are ordered w.r.t. their volume in ascending direction ($pvol \uparrow$). The patterns having the same volume are ordered w.r.t. their support in descending direction ($sub \downarrow$), and the patterns having the same volume and support are ordered lexicographically in ascending direction ($lex \uparrow$). The pseudocode of the cover algorithm is given in Algorithm 1. Given a set of patterns $\mathcal{H}$ the COVER algorithm arranges them in the standard cover order (line 2), and starts to cover gradually all yet uncovered objects $\mathcal{U}$. At each iteration of the loop (lines 4-9), COVER extracts the top pattern h from the list of patterns $\mathcal{H}^*$ arranged in the standard cover order. Then all uncovered objects included in h are declared as covered. These objects make a set $cover(h, G, \mathcal{H})$ and are removed from the set of uncovered objects.

Algorithm 1 COVER$(G, \mathcal{H})$

Input: object set G (i.e., data points in D^*),
pattern set $\mathcal{H}$
Output: cover of G by $\mathcal{H}$
1: $result \leftarrow \emptyset$
2: $\mathcal{H}^* \leftarrow StandardCoverOrder(\mathcal{H})$
3: $\mathcal{U} \leftarrow G$
4: **while** $(|\mathcal{U}| > 0)$**and**$(|\mathcal{H}^*| > 0)$ **do**
5: $h \leftarrow popTop(\mathcal{H}^*)$
6: $cover(h, G, \mathcal{H}) \leftarrow \{g \in \mathcal{U} \mid g \in h\}$
7: $result \leftarrow result \cup cover(h, G, \mathcal{H})$
8: $\mathcal{U} \leftarrow \mathcal{U} \setminus cover(h, G, \mathcal{H})$
9: **end while**
10: **return** $result$

Table 1 Patterns h_2, h_8, h_9, and h_{10} in the standard cover order

	lvol	*sup*	*coordinates*
h_2	log(1)	1	(0,4)
h_9	log(3)	5	(4,4)
h_8	log(3)	4	(4,7)
h_{10}	log(5)	3	(0,0)

Example 3 Let us consider the running example given in Fig. 1 and pattern set $\mathcal{H}^* = \{h_2, h_9, h_8, h_{10}\}$. Pattern h_2 is on top since it has the smallest volume, patterns h_9 and h_8 are located just after h_2 because they have the smallest volume among the remaining ones, h_9 is located before h_8 because it has larger support. The last pattern is h_{10}, since it has the largest volume. In this particular case we did not use the lexicographical order since all patterns differ by their volume or support. When there are two patterns with the same volume and support, they are lexicographically ordered. For example, if the considered patterns had the same volume and support, they would be ordered in the lexicographical order as follows: h_{10}, h_2, h_9, and h_8. Since $(0, 0) < (0, 4) < (4, 4) < (4, 7)$.

4.2.3 Main algorithm

MINT starts from elementary hyper-rectangles sequentially discover patterns that minimize the description length $L(D, \mathcal{H})$ by merging a pair of currently optimal patterns from $\mathcal{H}$. To compute a candidate pattern based on a pair of patterns, we introduce the *join* operator $\oplus$ which computes the join of two hyper-rectangles h_j and h_k. Then, given $h_j = \langle [v_i^{(l)}, v_i^{(u)})\rangle_{i\in\{1,\dots,|M|\}}$ and $h_k = \langle [w_i^{(l)}, w_i^{(u)})\rangle_{i\in\{1,\dots,|M|\}}$, the join $h_j \oplus h_k$ is given by the smallest hyper-rectangle containing h_j and h_k, i.e., $h_j \oplus h_k = \langle [\min(v_i^{(l)}, w_i^{(l)}), \max(v_i^{(u)}, w_i^{(u)}))\rangle_{i\in\{1,\dots,|M|\}}$.

The minimization of $L(D, \mathcal{H})$ consists in computing iteratively candidates using pairs of currently optimal patterns and selecting one that provides the largest gain in the total description length. For candidate $h_j \oplus h_k$, the length gain is given by:

$$
\begin{aligned}
&\Delta L(\mathcal{H}, D, h_j, h_k)\\
&\quad = L(D, \mathcal{H}) - L(D, \mathcal{H} \cup \{h_j \oplus h_k\} \setminus \{h_j, h_k\})\\
&\quad = \underbrace{L_{\mathbb{N}}(|\mathcal{H}|) - L_{\mathbb{N}}(|\mathcal{H}| - 1) + \left(\sum_{i=1}^{|M|} \log\left(|\mathcal{B}_i|(|\mathcal{B}_i| + 1)/2\right)\right)}_{\Delta L(\mathcal{H})}\\
&\qquad + \underbrace{\log \frac{\Gamma(|G| + \varepsilon|\mathcal{H}|)}{\Gamma(|G| + \varepsilon(|\mathcal{H}| - 1))} + \log \frac{\Gamma(\varepsilon(|\mathcal{H}| - 1))}{\Gamma(\varepsilon|\mathcal{H}|)} - \log \frac{\Gamma(usg(h_j) + \varepsilon)\Gamma(usg(h_k) + \varepsilon)}{\Gamma(usg(h_j \oplus h_k) + \varepsilon)\Gamma(\varepsilon)}}_{\Delta L(D(\mathcal{H}))}\\
&\qquad + \underbrace{usg(h_j)size(h_j) + usg(h_k)size(h_k) - usg(h_j \oplus h_k)size(h_j \oplus h_k)}_{\Delta L(D \ominus D(\mathcal{H}))}.
\end{aligned}
\tag{3}
$$

The term "gain" stands for the difference between the total description lengths obtained using the current pattern set $\mathcal{H}$ and the pattern set where patterns h_j and h_k are replaced by its join $h_j \oplus h_k$.

Since computing the gain in the total description length for each candidate $h_j \oplus h_k$ is computationally expensive (i.e., it requires recomputing the cover using Algorithm 1 for each candidates), we use an approximate *cover* to estimate ΔL. Then the approximate cover of a candidate $h_j \oplus h_k$ is defined as the union of the covers of h_j and h_k, i.e. $cover(h_j \oplus h_k, G) = cover(h_j, G) \cup cover(h_k, G)$. The cover of the remaining patterns from $\mathcal{H}$ does not change. As above, the usage of $h_j \oplus h_k$ is simply the cardinality of its cover, i.e., $usg(h_j \oplus h_k) = |cover(h_j \oplus h_k, G)|$. Since initially each object is covered by one pattern, in the new cover each object will be covered by one pattern as well. Finally the cover of the candidate with the highest gain is recomputed using Algorithm 1.

The pseudocode of MINT is given in Algorithm 2. At the beginning, the optimal patterns $\mathcal{H}$ are the elementary hyper-rectangles (line 2) induced by an equal-width discretization into a chosen number of intervals $|\mathcal{B}_i|$, $i = 1, \ldots, |M|$. We also set an additional parameter k to limit the number of candidates corresponding to k nearest neighbors. For large datasets and large number of intervals $|\mathcal{B}_i|$, setting a low value $k \ll |G|$ reduces the computational effort. In the discretized space, the elementary hyper-rectangles are points, thus the distance between them is the Euclidean distance. Then, given a hyper-rectangle $h_j \oplus h_k$, the neighbors are the neighbors of h_j and h_k. The main loop in lines 5-21 consists in selecting the best candidates from the current set of candidates $\mathcal{C}$, updating the set of optimal patterns $\mathcal{H}$, and collecting new candidates in $\mathcal{C}_{new}$. Once all candidates from $\mathcal{C}$ have been considered in the inner loop (lines 8-19), the new candidates from $\mathcal{C}_{new}$ become current ones (line 20). In the inner loop, lines 8-19, the candidates that minimize the total description length are selected one by one. They are considered by decreasing gain ΔL. At each iteration of the inner loop, the candidate $h_j \oplus h_k$ providing the largest gain ΔL is taken. New pattern set $\mathcal{H}^*$ includes $h_j \oplus h_k$ and does not contain h_j and h_k. We compute new cover by $\mathcal{H}^*$ using Algorithm 1 and then the total length $L(D, \mathcal{H}^*)$ (line 11). If $h_j \oplus h_k$ allows for a shorter total description length, patterns h_j and h_k are replaced with $h_j \oplus h_k$ in $\mathcal{H}$, and the candidates based on the newly added pattern are added to $\mathcal{C}_{new}$ (line 14) and are no more considered at the current iteration. In $\mathcal{C}_{new}$ we store pairs of indices making new candidates, and only in line 20 we compute candidates by calculating the gains ΔL that they provide. These gains, however, are computed only for patterns $h_j \oplus h_k$ where both h_j, h_k are still present in the set $\mathcal{H}$. Postponing the computation of candidates to line 20 allows us to reduce the number of candidates and to speed up pattern mining. The outer loop stops when there are no more candidates in $\mathcal{C}$.

Complexity of MINT. In the beginning, the number of candidates, i.e., pairs of elementary hyper-rectangles (line 4) is $O(min(|\mathcal{H}|^2, k \cdot |\mathcal{H}|))$, where $|\mathcal{H}|$ is the number of non-empty elementary hyper-rectangles. The number of elementary hyper-rectangles does not exceed the number of objects $|G|$. Thus, setting $k \ll |\mathcal{H}|$ we have the number of candidates $O(k \cdot |\mathcal{H}|) \sim (k \cdot |G|)$ which is linear w.r.t. the number of objects. Further, the number of hyper-rectangles can only decrease. Computing a candidate $h_j \oplus h_k$ takes $O(|M|)$. The number of candidates added to $\mathcal{C}_{new}$ at each iteration of the inner loop is equal to $|\mathcal{H}| - 2$, $|\mathcal{H}| - 3$, etc. Thus, the total number of candidates is

Algorithm 2 MINT(D^*, $|\mathcal{B}_i|$, k, N)

Input: numerical dataset D^*,
number of intervals for each attribute $|\mathcal{B}_i|$, $i = 1, \dots, |M|$,
number of the nearest neighbors for computing the candidates k
number of candidates for pruning N
Output: pattern set $\mathcal{H}$,
total description length $\mathcal{L}_{total}$

$D \leftarrow DiscretizeData(D^*, \{|\mathcal{B}_i| \mid i = 1, \dots, |M|\})$
$\mathcal{H} \leftarrow GetElementaryHyper\text{-}rectangles(D)$
$\mathcal{L}_{total} \leftarrow L(D, \mathcal{H})$
$\mathcal{C} \leftarrow \cup_{h_j \in \mathcal{H}, h_k \in NN(h_j,k)} (h_j \oplus h_k, \Delta L(\mathcal{H}, D, h_j, h_k))$
while $|\mathcal{C}| > 0$ **do**
  $\mathcal{C}_{new} \leftarrow \emptyset$
  $(h_j \oplus h_k, \Delta L) \leftarrow PopLargestGain(\mathcal{C})$
  while $\Delta L > 0$ **do**
    if $h_j, h_k \in \mathcal{H}$ **then**
      $\mathcal{H}^* \leftarrow \mathcal{H} \cup \{h_j \oplus h_k\} \setminus \{h_j, h_k\}$
      $\mathcal{L}_{new} \leftarrow L(D, \mathcal{H}^*)$ (GMINT: $\mathcal{L}_{new} \leftarrow L_{total} - \Delta L$)
      if $\mathcal{L}_{new} \leq \mathcal{L}_{total}$ **then**
        $\mathcal{H} \leftarrow \mathcal{H}^*$
        $\mathcal{C}_{new} \leftarrow \mathcal{C}_{new} \cup \{(h_j \oplus h_k) \oplus h \mid h \in \mathcal{H}, h \neq h_j \oplus h_k\}$
        $\mathcal{L}_{total} \leftarrow \mathcal{L}_{new}$
      end if
    end if
    $(h_j \oplus h_k, \Delta L) \leftarrow PopLargestGain(\mathcal{C})$
  end while
  $\mathcal{C} \leftarrow \mathcal{C}_{new}$
end while
return $\mathcal{H}, \mathcal{L}_{total}$

$O(|\mathcal{H}|^2)$. The total complexity of computing candidates is $O(|M| \cdot |\mathcal{H}|^2)$. Searching for the maximal gain among the candidates takes $O(|\mathcal{H}|^2)$ time. Since there can be at most $\mathcal{H}$ iterations, computing the maximal gain takes $O(|\mathcal{H}|^3)$. When adding a candidate to $\mathcal{H}$ we need to maintain the standard cover order of hyper-rectangles, that takes $O(\log |\mathcal{H}|)$. Moreover, recomputing the cover takes in the worse case $O(|\mathcal{H}||G|)$. Since the maximal number of candidates is $|\mathcal{H}|^2$, the cover recomputing increases the complexity of MINT by $O\big(|\mathcal{H}|^2(\log |\mathcal{H}| + |\mathcal{H}||G|)\big)$.

In the worst case, where $|\mathcal{H}| = |G|$, the complexity is $O(|G|^2(|M|+|G|+log|G|+|G|^2)) \sim O(|G|^2|M| + |G|^4)$, however it is possible either to set a small number of initial intervals $|\mathcal{B}_i|$ ensuring $|\mathcal{H}| \ll |G|$ or to restrict the number of candidates.

Example 4 Let us consider how the algorithm works on the running example from Fig. 1.

Initially, the set of hyper-rectangles consists of elementary ones, i.e., $h_1, \dots, h_7$ (Fig. 2). We restrict the set of candidates by considering only 2 nearest neighbor for each pattern (they are given in Fig. 2, left). Thus, the set of candidates is given by $\mathcal{C} = \{h_1 \oplus h_2, h_1 \oplus h_3, h_2 \oplus h_4, h_3 \oplus h_4, h_4 \oplus h_5, h_4 \oplus h_6, h_5 \oplus h_7, h_6 \oplus h_7\}$. In the cases, when the number of equidistant nearest neighbor is greater than k, we select k of them with the smallest indices.

The patterns are added in the following order: $h_8 = h_5 \oplus h_7$, $h_9 = h_4 \oplus h_6$, $h_{10} = h_1 \oplus h_3$, which corresponds to decreasing gain.

pattern	coordinates		2 nearest neighbors
h_1	0	0	h_2, h_3
h_2	0	4	h_1, h_4
h_3	4	0	h_1, h_4
h_4	4	4	h_5, h_6
h_5	4	7	h_4, h_7
h_6	6	4	h_4, h_7
h_7	6	7	h_5, h_6

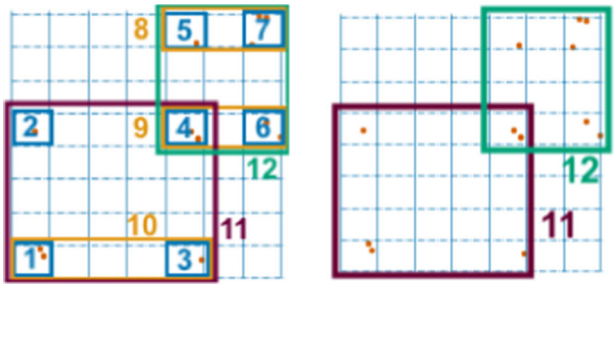


Fig. 2 The sequential changes in the set of currently optimal patterns. The initial set of hyper-patterns is composed of elementary ones, i.e., $h_1, \ldots, h_7$

After that, set $\mathcal{C}$ does not contain candidates that can improve the total length. Thus, MINT proceeds by considering the candidates from $\mathcal{C}_{new} = \{h_j \oplus h_k \mid j, k = 2, 8, 9, 10, j \neq k\}$, which are the candidates composed of pairs of recently added patterns and the unused old ones. The newly added patterns are $h_{11} = h_8 \oplus h_9$ and $h_{12} = h_2 \oplus h_{10}$. The new candidate set is $\mathcal{C}_{new} = \{h_{11} \oplus h_{12}\}$. The algorithm terminates, the set of hyper-rectangles corresponding to the smallest total description length is $\mathcal{H} = \{h_{11}, h_{12}\}$.

4.2.4 GREEDY MINT (GMINT)

The MINT algorithm can be not efficient in some situations because it requires recomputing the cover each time a new pattern enters the set $\mathcal{H}$. For reducing the computational time and reaching a better efficiency, the cover strategy proposed in § 4.2.2 and implemented in Algorithm 1 is approximated by using the same cover as in the estimate of ΔL, i.e., the cover of $h_j \oplus h_k$ is given by $cover(h_j \oplus h_k, G) = cover(h_j, G) \cup cover(h_k, G)$, and it remains unchanged for the other patterns in $\mathcal{H}$. The latter means that line 11 of Algorithm 2 is replaced with $\mathcal{L}_{new} = \mathcal{L}_{total} - \Delta L$. This modification gives rise to the GREEDY MINT (GMINT) version of MINT and is the only difference between the both versions. Indeed, GMINT does not recompute the covering but adjusts the cover only locally for the newly added patterns.

In terms of complexity, based on this optimized strategy, we get rid in the GMINT version of the heaviest component of the complexity term in MINT, which is $O(|G|^2(log|G| + |G|^2))$. Then the total time complexity is $O(|G|^2(|M| + |G|))$. *Comparison of complexity of* GMINT *with other miners.* The theoretical complexity of the algorithms related to (G)MINT algorithms is estimated based on different features. In REALKRIMP, the size of the sampling s mainly affects the time complexity, the cost of computing the first hyper-rectangle is $O(|M|s^2 \log s + |M||G|s)$, additional ones are mined in $O(|M|s^2 + |M||G|s)$. In the worst case, where the sample size is proportional to the number of objects, the time complexity is $O(|M||G| \log |G|)$ and $O(|M||G|^2)$ for the first and additional hyper-rectangles, respectively. In MINT the main component is the number of hyper-rectangles which is at most $|G|$, thus, the total time complexity

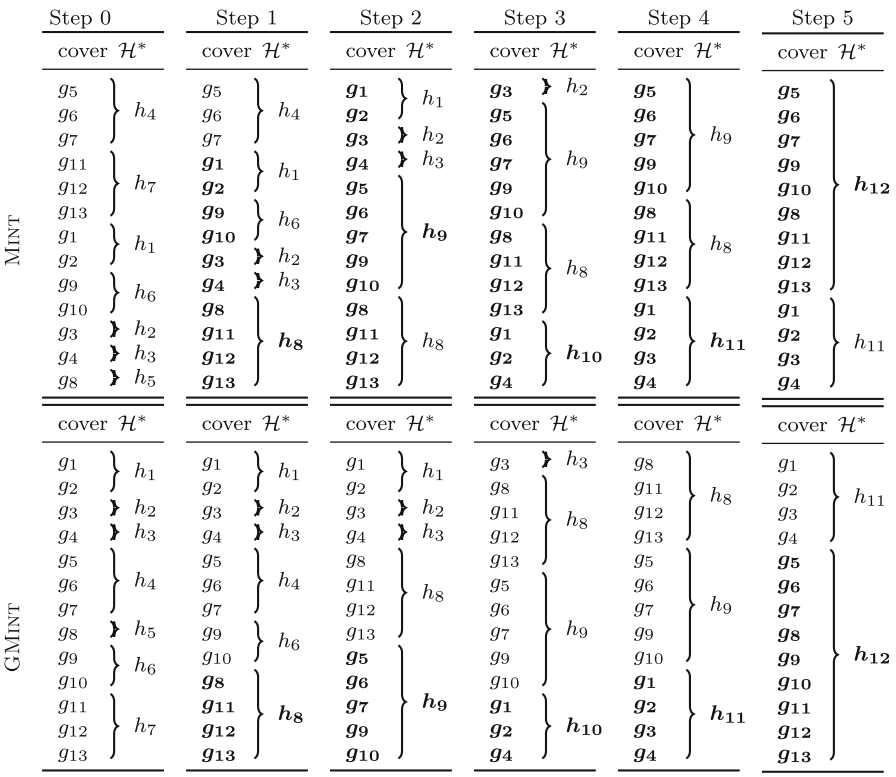


Fig. 3 Step-wise computing of the cover by MINT and GMINT. The objects $\boldsymbol{g_i}$ for which the cover is recomputed and added patterns $\boldsymbol{h_i}$ are highlighted in bold

is $O(|M||G|^2 + |G|^3)$. Thus, both REALKRIMP and SLIM have polynomial complexity w.r.t. the dataset size. SLIM has the largest worst-case complexity $O(|\mathcal{C}|^3|G||I|)$, where $|C| = O(2^{\min(|G|,|I|)})$ is the number of the candidates, which can be exponential in the size of the dataset. Moreover, the size of the dataset used in SLIM is larger than the size of the dataset used by MINT and REALKRIMP, since the number of attributes $|I|$ in a binarized dataset is larger than the number of attributes $|M|$ in the discretized one. Thus, REALKRIMP and (G)MINT have polynomial complexity in the input size. However, in practice, GMINT works much faster as shown in the experiments.

Example 5 Let us consider the difference in the cover strategy of MINT and GMINT. According to the cover strategy used in MINT (§ 4.2.2), a newly added pattern is inserted in the list of patterns ordered w.r.t. the standard cover order. After insertion, the cover of data is recomputed starting from the first altered position in the pattern list. Depending on the position of insertion, there will be more recomputing if the pattern list is altered at the top and less recomputing if the list is altered at the bottom. It appears that recomputing is much more important for MINT than for GMINT. In Fig. 3, we show the order of pattern insertion used by both algorithms to cover data. In bold we indicate the objects for which the cover is recomputed at each step and

patterns which are added. As we can see, in both cases the hyper-rectangles cover the same objects, however MINT requires much more computational efforts. For example, at Step 1 in MINT, when h_8 is added, we need to recompute the cover for all patterns except for h_4 (which is at the top of the list), because the list is changed from the 2nd position. Meanwhile in GMINT the cover is recomputed only for objects covered by h_5 and h_7, since they constitute a new pattern. In Appendix C we show in experiments that both strategies return almost the same pattern sets with almost the same compression quality, but MINT requires much more time.

5 Experiments

In this section, we compare MINT with SLIM and REALKRIMP –the most similar MDL-based approaches to numerical pattern mining.

SLIM and REALKRIMP are not fully comparable with our approach since they have their own parameters that actually affect both the performance and the quality of the results. SLIM works with binarized data, while MINT and REALKRIMP work with discretized data. However, SLIM and MINT allow for choosing the number of discretization intervals. SLIM is better adapted to mine patterns in datasets with a coarse discretization. A coarse discretization results in a moderate increase in the number of attributes, while a fine discretization usually results in a drastic increase and can make the task intractable. By contrast, MINT is able to mine patterns efficiently even in datasets with fine discretization. A last difference is that SLIM and MINT evaluate a pattern set as a whole, while REALKRIMP evaluates each pattern in a set independently.

In the experiments we report the results for GMINT, as MINT provides the same results as GMINT but requires more time. Moreover, MINT and GMINT are qualitatively and quantitatively compared in Appendix C.

5.1 Datasets

We selected datasets from the UCI repository (Dua and Graff 2017) that are generally used in itemset mining, and discretized with the LUCS-KDD DN software (Coenen 2003). Since MINT implicitly discretizes datasets into equal-width intervals, we were compelled to use the same data discretization for other algorithms. The choice of the optimal number of intervals depends both on the chosen algorithm and the dataset. In experiments we do not tune the parameters specifically for each dataset and each algorithm. We considered the parameters that work generally well for the considered algorithms, namely splitting into 5 intervals (as in LUCS-KDD repository, where the number of intervals is limited to 5 for each attribute) and into $\sqrt{|G|}$.

Since the proposed discretization into 5 intervals may be non-optimal, we also use IPD discretization, which is expected to provide only necessary discretization intervals, thus should be the most suitable for SLIM. Moreover, as SLIM deals with binary data, we additionally transform each discretization interval into a binary attribute. The parameters of the real-world datasets are given in Table 2.

Table 2 Description of datasets and the parameters of the discretization (applicable to MINT and SLIM) and the compression ratio of MINT. All attributes are numerical

data parameters							MINT CR, %		
name	$\|G\|$	$\|M\|$	#intervals			classes	#intervals		
			5	$\sqrt{\|G\|}$	IPD		5	$\sqrt{\|G\|}$	IPD
iris	150	4	20	45	8	3	69	42	97
wine	178	13	65	163	26	3	46	46	56
haberman	306	3	15	42	6	2	75	48	97
ecoli	336	7	29	91	14	8	49	33	90
breast wisconsin	569	30	146	581	84	2	31	37	43
spambase	4601	57	266	1794	149	2	29	18	41
waveform	5000	40	200	2681	198	3	54	49	55
parkinsons	5875	18	87	961	92	-	52	27	39
statlog satimage	6435	36	180	2335	213	6	31	37	33
gas sensor	13910	128	566	8915	1060	27	19	15	19
avila	20867	10	37	445	88	12	95	22	42
credit card	30000	24	112	1401	195	2	36	24	39
shuttle	58000	8	40	459	91	77	99	70	55
sensorless dd	58509	48	219	5022	693	11	50	21	37
mini	130064	50	117	1623	1031	2	91	9	47
workloads	200000	5	25	2109	296107	-	99	40	46

5.2 Parameters of the algorithms

The discretization described above (into 5, $\sqrt{|G|}$ itervals) is performed in MINT implicitly, thus is set as a parameter. We set the number of neighbors, considered for computing candidates, to be equal to the number of intervals. The performance of MINT with other parameters is reported in Appendix E.

REALKRIMP relies on a set of parameters, the most important among them are *sample size*, *perseverance*, and *thoroughness*. Sample size defines the size of the dataset sample that will be used to compute patterns. We consider samples of size $\sqrt{|G|}$, $0.25|G|$ and $0.5|G|$. The sample size affects the running time: the smaller samples allow for faster pattern mining operations, while too small samples may prevent to discover interesting patterns. Perseverance regulates the behavior of REALKRIMP to reach a global minimum of the description length. Large values of perseverance help to reach a global minimum. Perseverance is then set to 1/5 of the sample size. Thoroughness is the maximal number of consecutive non-compressible hyper-intervals that should be considered. We set thoroughness equal to 100.

5.3 Compression ratio

Firstly, we consider the main quality measure of the MDL-based methods, namely *compression ratio*. Since MINT is not comparable to other pattern mining approaches,

namely SLIM and REALKRIMP, we report only the compression ratio of MINT in Table 2 (the lower values, the better). MINT provides the best compression for the data discretized into $\sqrt{|G|}$ intervals. High compression ratio (worse compression) for data discretized into 5 intervals is quite expected, since this coarse discretization provides large elementary hyper-rectangles that cannot be compressed efficiently. For IPD discretized data, high compression ratios may indicate that the IPD-discretization grid splits the space into hyper-rectangles that MINT considers as separate patterns and does not compress further, while low compression ratios indicate the cases where IPD returns hyper-rectangles that can be further efficiently compressed by MINT.

We report the compression ratios for other methods and discuss the issues related to their incomparability in Appendix D.

5.4 Running time

The running time is reported in Table 3. The cases where pattern sets are not computed are indicated in the table with "...". The performance of SLIM and MINT is affected by the number of discretization intervals, while the performance of REALKRIMP depends heavily on the sample size.

The running time show that for small datasets ($< 1k$) all methods work fast and often complete the work in less than a second. For average-sized ($< 50k$) and large ($> 50k$) datasets the running time depends on the chosen parameters. SLIM mines fast patterns in datasets with a coarse discretization (into 5 intervals). However, for fine discretizations (into $\sqrt{|G|}$ or IPD) the running time increases drastically. For example, for "shuttle" dataset, SLIM terminates in 1 second, when the number of intervals is 5, while when the attribute ranges are split into $\sqrt{|G|}$ intervals and by IPD, SLIM requires 17394 and 650 seconds, respectively. Again for SLIM, the scalability issues are especially pronounced for datasets with a large number of attributes, e.g., for "gas sensor" dataset, SLIM terminates in 8206, ...,[1] and 72488 seconds, while MINT requires only 5, 956, and 54 seconds for the same discretization parameters.

REALKRIMP also suffers from poor scalability: for average- or large-sized datasets, setting a small sample size, e.g., $\sqrt{|G|}$, does not allow to find a sufficient amount of interesting patterns, while setting a reasonable sample size ($0.25|G|$ or $0.5|G|$) results in a drastic increase of the running time and memory requirement.

Our experiments show that MINT, dealing with the same discretization as SLIM, requires less time to mine patterns, especially for large datasets. However, it is not enough to assess the performance of the patterns by considering their running time. It is also important to study how many patterns they return and which kind of patterns.

5.5 Number of MDL-selected patterns

Intuitively, the number of patterns should be small but enough sufficient to describe all interesting relations between attributes. The number of MDL-selected patterns for the studied methods is reported in Table 4. The table shows that, given the same

[1] The experiments are not completed within one week.

Table 3 Running time, in seconds, of SLIM and MINT and REALKRIMP

name	$\|G\|$	$\|M\|$	SLIM			MINT			REALKRIMP		
			# intervals			# intervals			sample size		
			5	$\sqrt{\|G\|}$	IPD	5	$\sqrt{\|G\|}$	IPD	$\sqrt{\|G\|}$	$0.25\|G\|$	$0.5\|G\|$
iris	150	4	0	0	0	0	0	0	0	0	0
wine	178	13	1	0	0	0	0	0	0	0	0
haberman	306	3	0	0	0	0	0	0	0	0	0
ecoli	336	7	0	0	0	0	0	0	0	1	1
breast w.	569	30	3	53	0	1	1	3	0	7	22
spambase	4601	57	11	6159	63	1	49	6	6	14169	18133
waveform	5000	40	505	50022	1346	16	149	21	2	5510	11940
parkinsons	5875	18	3	599	27	1	79	7	6	999	1515
statlog s.	6435	36	790	32212	1876	13	207	10	8	2849	7140
gas sensor	13910	128	8206	...[a]	72488	5	956	54	113	42677	136802
avila	20867	10	1	4395	79	0	5727	244	69	8854	28855
credit card	30000	24	38	29422	17083	38	15588	473	171	192057	410780
shuttle	58000	8	1	17394	650	0	90	82	200	...[b]	...[b]
sensorless	58509	48	60	500263	83819	4	133824	1433	508	299237	...[b]
mini	130064	50	22	...[a]	...[a]	0	55889	8074	861	...[b]	...[b]
workloads	200000	5	27	113195	8177	0	128057	5089	1031	...[b]	...[b]

[a] Interrupted process. Experiments are not completed after one week.
[b] Interrupted process because of the lack of memory (requires more than 64GB)

discretization, SLIM returns usually a larger number of patterns than MINT. As with the running time, SLIM is sensitive to a large number of attributes and in this case usually returns a much larger number of patterns than MINT. For example, for "gas sensor" dataset SLIM returns 1608, ...,[2] and 9554 patterns, while MINT, with the same discretization settings, returns only 306, 571 and 897 patterns, respectively.

REALKRIMP, on the contrary, returns a much smaller number of patterns than MINT and SLIM. For example, for "gas sensor" dataset it returns only 4, 30, and 49 patterns for samples of size $\sqrt{|G|}$, $0.25|G|$, and $0.5|G|$, respectively. Taking into account the running time, we can conclude that with the chosen parameters, the average running time per pattern is much larger for REALKRIMP than for SLIM and MINT. Thus, REALKRIMP has the highest "cost" in seconds of generating a pattern.

Now, let us examine the quality of the generated patterns.

5.6 Pattern similarity (redundancy)

Pattern similarity is particularly important for REALKRIMP, where patterns are mined w.r.t. other patterns, but evaluated independently, and there are no guarantee of avoiding the selection of very similar patterns. To study pattern similarity, we consider the average pairwise Jaccard similarity computed w.r.t. the sets of objects that the patterns describe. We take into account all occurrences of patterns in data rather than their usage in the data cover (which is, by definition, non-redundant).

However, the average pairwise Jaccard similarity applied to a large pattern set may not be able to spot "local" redundancy, i.e., when there is a small subsets of almost the same patterns. To tackle this issue, we consider instead for each pattern we select at most 10 patterns among the most similar patterns w.r.t. Jaccard similarity and report the average value (removing the repetitive pairs if they appear). The average values of similarity are presented in Fig. 4a.

The results of the experiments show that on average the pairwise Jaccard similarity is the smallest for MINT, and only slightly higher for SLIM. In SLIM higher values are caused by the fact that each object can be covered by different non-overlapping itemsets, thus these increased values of the Jaccard similarity are partially caused by the specificity of the model. REALKRIMP has the largest values of the Jaccard similarity, close to 1 (see Fig. 4a). This result is quite expected since the patterns are evaluated independently, thus the method does not minimize redundancy in the pattern set.

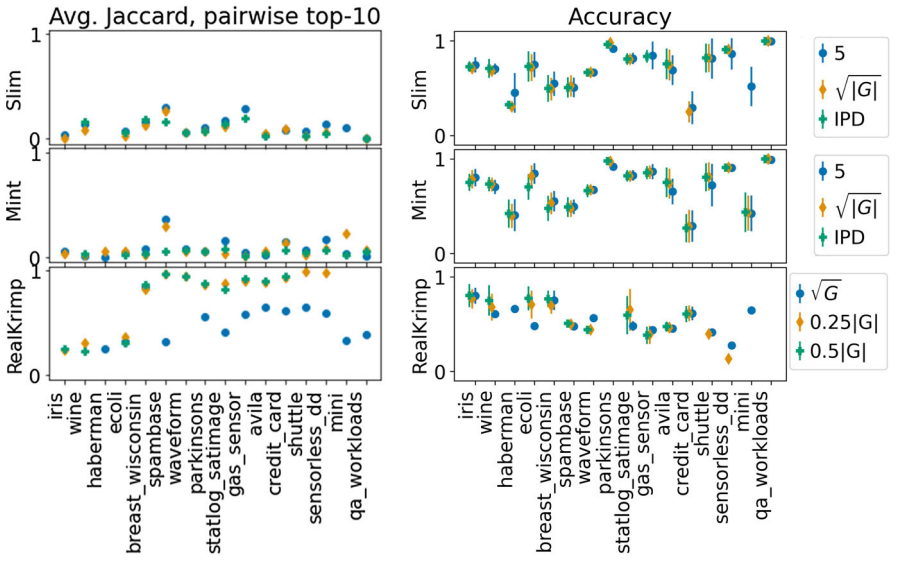
5.7 Purity of patterns

To evaluate the significance of the resulting patterns, we measure their purity by considering the classes of objects they describe. The class labels are not used during pattern mining and are considered only for assessing the pattern quality. We assign to a pattern a label of the majority class of objects described by it. In Fig. 4b we show the accuracy patterns computed based on objects the pattern describes. The results show that SLIM and MINT, being based on the same discretizations, have quite similar

[2] The experiments are not completed within one week.

Table 4 The number of MDL-selected patterns by SLIM and MINT for discretization into 5, $\sqrt{|G|}$ intervals and the IPD discretization

name	$\|G\|$	$\|M\|$	SLIM			MINT			REALKRIMP		
			# intervals			# intervals			sample size		
			5	$\sqrt{\|G\|}$	IPD	5	$\sqrt{\|G\|}$	IPD	$\sqrt{\|G\|}$	$0.25\|G\|$	$0.5\|G\|$
iris	150	4	19	18	9	8	8	6	2	4	6
wine	178	13	85	62	42	24	16	17	1	4	9
haberman	306	3	13	12	8	7	11	3	2	0	0
ecoli	336	7	50	47	22	19	11	12	2	7	12
breast w.	569	30	246	564	219	52	33	66	19	12	23
spambase	4601	57	301	1486	881	83	201	504	2	95	97
waveform	5000	40	1726	4071	1645	1177	148	73	1	65	199
parkinsons	5875	18	256	1418	708	107	209	404	4	21	25
statlog s.	6435	36	1322	6480	1871	716	260	457	3	32	41
gas sensor	13910	128	1608	...	9554	306	571	897	4	30	49
avila	20867	10	135	2149	1187	53	624	1478	8	23	33
credit card	30000	24	733	4002	3757	1341	1233	2226	9	115	189
shuttle	58000	8	57	2947	1530	21	968	1723	5	0	0
sensorless	58509	48	571	14040	10466	500	1310	2669	4	27	0
mini	130064	50	150	...	...	42	926	8319	2	0	0
workloads	200000	5	688	4773	5889	318	3273	5061	8	0	0



(a) Jaccard similarity computed on the top-10 Jaccard-similar pairs for each pattern, repetitive pairs are discarded

(b) Accuracy (purity) of patterns

Fig. 4 The average values of quality measures

average accuracy. REALKRIMP return patterns with high accuracy for small datasets, however, loses in accuracy on large datasets.

5.8 Accuracy of pattern descriptions

As it was mentioned in the introduction, in pattern mining it is important not only to describe meaningful groups of objects but also to provide quite precise boundaries of these groups. Unfortunately, we cannot evaluate how precise are the pattern boundaries for the real-world datasets since we do not have any ground truth patterns.

To evaluate the precision of pattern boundaries of patterns we use synthetic datasets. We generate 6 types of 2-dimensional datasets with different numbers of patterns and different positions of patterns w.r.t. other patterns, shown in Fig. 5. The detailed parameters of the synthetic datasets are given in the extended version of the paper. The ground truth patterns are highlighted in different colors. Further, we use $\mathcal{T}$ to denote a set of ground truth hyper-rectangles. For all these types of data, we generate datasets where each pattern contains 100, 200, 500, 700, 1000 objects.

In Fig. 5, the "simple" datasets consist of separable patterns. The "variations" datasets contain adjacent patterns and thus allow for variations in pattern boundaries. The "inverted" datasets include the most complicated patterns for MINT and SLIM since they treat asymmetrically dense and sparse regions. It means that these algorithms are not able to identify the hole in the middle. Instead of this hole, we may expect a complicated description of the dense region around this hole. "Simple overlaps"

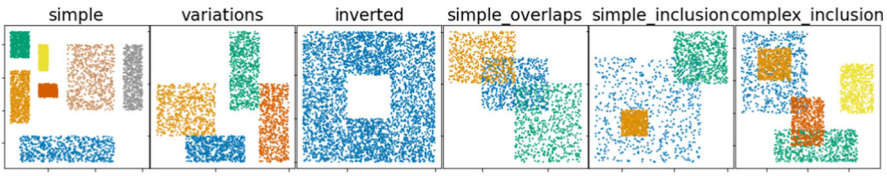


Fig. 5 Six types of generated synthetic dataset

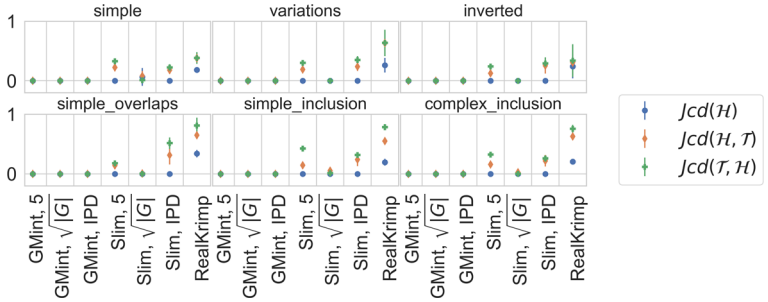


Fig. 6 Average Jaccard similarity of hyper-rectangles

contains overlapping patterns, while "simple inclusion" and "complex inclusion" can also contain patterns that are subsets of other patterns.

For synthetic datasets we use the same settings as for real-world datasets, i.e., discretization into 5, $\sqrt{|G|}$ intervals, and IPD discretization (for MINT and SLIM) and the default settings from (Witteveen 2012) for REALKRIMP.

We evaluate the quality of patterns using the Jaccard similarity applied to hyper-rectangles. For two hyper-rectangles h_1 and h_2 the Jaccard similarity is given by $Jaccard(h_1, h_2) = area(h_1 \cap h_2)/area(h_1 \oplus h_2)$, where $h_1 \cap h_2$ and $h_1 \oplus h_2$ is the intersection and join of h_1 and h_2, respectively.

We begin with the average pairwise Jaccard similarity of the computed patterns:

$$Jcd(\mathcal{H}) = \frac{\sum_{h_i, h_j \in \mathcal{H}, i \neq j} Jaccard(h_1, h_2)}{0.5|\mathcal{H}|(|\mathcal{H}| - 1)}.$$

The values reported in Fig. 6 show that SLIM returns non-redundant patterns since they are non-overlapping, while REALKRIMP returns very similar patterns. These patterns are redundant since even for the "simple" datasets, where all ground truth patterns are separable and non-overlapping, the patterns returned by REALKRIMP are very similar. The similarity of MINT-selected patterns is very low, but it increases for the datasets with overlapping patterns, e.g., "simple overlaps" or "simple inclusion", and is almost 0 for the datasets with non-overlapping patterns, e.g., "simple" or "variations".

The next question is how well the boundaries of the computed patterns from $\mathcal{H}$ are aligned with the boundaries of the ground truth patterns from $\mathcal{T}$, i.e., the patterns that we generated.

To compute Jaccard similarity between two different pattern sets $\mathcal{H}_1$ and $\mathcal{H}_2$ we take into account for each $h_1 \in \mathcal{H}_1$ only the most Jaccard-similar pattern $h_2 \in \mathcal{H}_2$:

$$Jcd(\mathcal{H}_1, \mathcal{H}_2) = \frac{\sum_{h_1 \in \mathcal{H}_1} \max_{h_2 \in \mathcal{H}_2} Jaccard(h_1, h_2)}{|\mathcal{H}_1|}. \quad (4)$$

The idea behind this measure is to assess how similar the "matched" pairs from $\mathcal{H}_1$ and $\mathcal{H}_2$. This measure asymmetric.

In Fig. 6 we show the average values of $Jcd(\mathcal{H}, \mathcal{T})$ and $Jcd(\mathcal{T}, \mathcal{H})$. The values of $Jcd(\mathcal{H}, \mathcal{T})$ close to 1 indicate that all computed patterns are very similar to patterns given in ground truth. The worst results corresponds to SLIM: even with a "smart" IPD discretization, the boundaries computed by SLIM are not very precise. The low values for fine discretized data are explained by the inability of SLIM to merge elementary hyper-rectangles.

MINT with IPD-discretization returns also quite poor results. However, in the best settings (discretization into $\sqrt{|G|}$ intervals), MINT and REALKRIMP have quite high values of $Jcd(\mathcal{H}, \mathcal{T})$. The latter means that all patterns from $\mathcal{H}$ are quite similar to the patterns given by ground truth.

The values $Jcd(\mathcal{T}, \mathcal{H})$ close to 1 correspond to the case where each ground truth pattern has at least one pattern in $\mathcal{H}$ that is very similar to ground truth patterns. The results show that the quality of MINT-generated patterns for the default discretization into $\sqrt{|G|}$ intervals is the best. For some datasets REALKRIMP works equally well, e.g., "simple overlaps" or "simple inclusion", but for others, it may provide quite bad results, e.g., for the simplest set of patterns contained in the "simple" datasets.

Comparing the values of $Jcd(\mathcal{H}, \mathcal{T})$ and $Jcd(\mathcal{T}, \mathcal{H})$ we may conclude that REALKRIMP returns a lot of similar patterns, but these patterns do not match the ground truth patterns as well as the patterns generated by MINT. Thus, the experiments show that MINT (with a fine discretization) returns patterns with quite precise boundaries and outperforms the state-of-the-art MDL-based pattern miners.

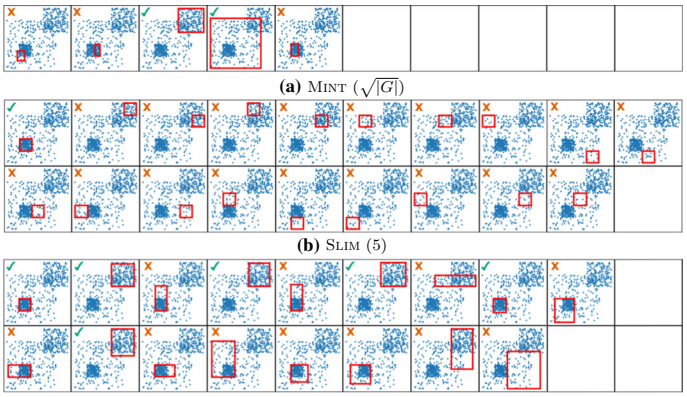
6 Visualization of some hyper-rectangles for synthetic data

Despite the fact that Jaccard similarity describes well the quality of patterns $\mathcal{H}$ and their alignments with $\mathcal{T}$, it does not allow to examine patterns entirely. In Fig. 7 we show the patterns computed by SLIM, REALKRIMP, and MINT, when applied to the "simple inclusion" dataset with 3 ground truth patterns having support equal to 200. The patterns for other datasets, due to limited space, are reported in the extended version of the paper.[3]

As it can be seen, SLIM returns patterns that are completely determined by the chosen discretization. This is a typical limitation of itemset mining algorithms when applied to numerical data.

MINT returns 5 patterns. Among them two patterns that correspond exactly to the ground truth patterns and the remaining three patterns describe the smallest ground

[3] arxiv.org/abs/2011.14843.



(a) MINT ($\sqrt{|G|}$)

(b) SLIM (5)

(c) REALKRIMP (sampling of size 0.5)

Fig. 7 The results of pattern mining for "Simple inclusion" dataset, support of the ground truth patterns is 200

truth pattern. Nevertheless, their join gives a quite correct pattern. REALKRIMP distinguishes only 2 patterns correctly and at the same time it returns a lot of similar patterns. Indeed the number of patterns discovered by REALKRIMP is 18 while the number of the ground truth patterns is only 3. Then it is quite hard for REALKRIMP to find a good combination of patterns allowing the reconstruction of the third pattern. It can be concluded that redundant patterns are raising a more important problem for REALKRIMP than for MINT.

For all other datasets we observe quite the same behavior. SLIM returns very imprecise patterns, whose quality depends on the quality of the discretization grid. REALKRIMP returns well-aligned patterns within a large number of very similar patterns. By contrast, MINT returns a reasonable number of patterns, which are less redundant than those returned by REALKRIMP, and usually quite well aligned with the ground truth patterns.

7 Discussion and conclusion

In this paper we propose a formalization of numerical pattern set mining problem based on MDL principle and we focus on the following characteristics: (i) interpretability of patterns, (ii) precise pattern descriptions, (iii) non-redundancy of pattern sets, and (iv) scalability. In the paper we study and materialize these characteristics, and we also propose a working implementation within a system called MINT.

By "interpretability" we mean not only the ability to explain why a particular pattern is selected, but also the ease of analyzing a set of discovered numerical patterns for a human agent. With this regard, patterns of arbitrary shapes or even polygons may be not an appropriate choice when considering multidimensional numerical data. This is why we decided to work with one of the most common shapes, namely "hyper-rectangles", which are currently used in numerical pattern mining and related tasks.

Another important requirement is that the boundaries of patterns should be "well-defined" and "quite precise". By contrast, a common approach to numerical pattern mining consists in data discretization and binarization followed by reduction to itemset mining. Such an approach suffers from various drawbacks among which (i) the boundaries of patterns are not well-defined and this heavily affects the output, (ii) the scalability is not good because of the potential exponential number of attributes due to scaling, (iii) the information loss related to the loss of the interval order within a range may be very important.

In our experiments we compare the behavior of MINT with the MDL-based itemset set miner SLIM (associated with a scaling of numerical data). The experiments demonstrate that SLIM generally provides quite poor patterns. Actually, when the discretization is too fine, SLIM is not able to merge patterns into larger patterns, while when the discretization is too coarse the algorithm returns very imprecise boundaries. In addition, we also consider another MDL-based algorithm, namely REALKRIMP, which is, to the best of our knowledge, the only MDL-based approach dealing with numerical pattern mining without any prior data transformation. However, one main limitation of REALKRIMP is that it mines patterns individually and then the resulting patterns are very redundant.

Furthermore, in the experiments, both REALKRIMP and SLIM show a poor scalability. MINT may also have a high running time for some large datasets, but in staying still at a reasonable level. MINT may be correlated with IPD –for "Interaction-Preserving Discretization"– but both systems perform different tasks. MINT could work in combination with IPD since the latter does not return exactly patterns but mainly MDL-selected boundaries. The elementary hyper-intervals induced from IPD results are only fragments of ground truth patterns. Then MINT could be applied to merge these elementary hyper-rectangles into larger hyper-rectangles.

Indeed, our experiments show that the data compressed by IPD can be even more compressed in applying MINT, i.e., the patterns as computed by IPD should still be completed for being comparable to those discovered by MINT. However, as the experiments show, directly applying MINT to fine discretized data allows to obtain better results than applying IPD as a preprocessing step. This can be explained by the fact that IPD returns uniform or global boundaries, which are less precise than the boundaries specifically "tuned" by MINT for each pattern.

For summarizing, the MINT algorithm shows various very good capabilities w.r.t. its competitors, among which a good behavior on fine discretized datasets, a good scalability, and an output of a moderate including non-redundant patterns with precise boundaries. However, there is still room for improving MINT, for example in avoiding redundant patterns and in the capability of mining sparse regions in the same manner as dense ones.

Future work may be followed in several directions. Here, MINT works with an encoding based on prequential plug-in codes. It could be interesting to reuse another encoding and to check how the performance of the system evolve, trying to measure what is the influence of the encoding choice. Moreover, we should consider more datasets and especially large and complex datasets, and try to measure the limit of the applicability of MINT, for in turn improving the algorithm in the details. In general, more experiments should still be considered for improving the quality of MINT.

Another interesting future direction is to use MINT in conjunction with a clustering algorithm. This could be a good way of associating descriptions or patterns with classes of individuals discovered by a clustering process. In this way a description in terms of attribute and ranges of values could be attached to the discovered clusters and complete the definition of the set of individuals which are covered. This could be reused in ontology engineering for example, and as well in numerous tasks where clustering is heavily used at the moment.

Acknowledgements Sergei O. Kuznetsov gratefully acknowledges the support from the Basic Research Program of the National Research University Higher School of Economics.

Appendix A Derivation of the plug-in codes

Let H^n be the sequence $h^1, \ldots, h^{n-1}, h^n$. The idea of the prequential codes is to assess the probability of observing the n-th element h^n of the sequence H^n based on the previous elements $h^1, \ldots, h^{n-1}$: $P_{plug\text{-}in}(h^n) = \prod_{i=1}^{n} P(h^i|h^{i-1})$.

Initially, a uniform distribution over $\mathcal{H}$ is defined with a pseudo-count ε over the set of patterns $\mathcal{H}$, i.e., the probability of h^0 is given by $P(h^0) = \frac{\varepsilon}{\varepsilon \cdot |\mathcal{H}|}$. Then, during the process of transmitting/receiving messages, the pattern probabilities and lengths are updated w.r.t. the patterns observed so far.

At each single step, the distribution P over $\mathcal{H}$ is multinomial with parameters $(\theta_1, \ldots, \theta_{|\mathcal{H}|})$, where θ_i corresponds to a pattern $h_i \in \mathcal{H}$. With the subscript indices we arbitrarily enumerate patterns in $\mathcal{H}$. The superscript indices denote the sequence number of patterns in the transmitting sequence. Further, we will see that the order of patterns in the sequence $h^1, \ldots, h^n$ does not affect the length of the encoded sequence. This length depends only on the number of times each pattern appears in the sequence.

Taking into account the initial probabilities, the maximum-likelihood estimates of the parameters of the multinomial distribution, given the sequence $H^n = h^1 \ldots h^n$, are the following:

$$\hat{\theta}_{h^n}(h) = \frac{usg(h|h^1 \ldots h^n) + \varepsilon}{\sum_{h^* \in \mathcal{H}} usg(h^*|h^1 \ldots h^n) + \varepsilon|\mathcal{H}|}, \tag{5}$$

where $usg(h|h^1 \ldots h^n)$ is the number of occurrences of pattern h in the sequences observed so far (up to the n-th pattern included). The maximum likelihood (ML) estimates from Equation 5 are equivalent to the probability estimates of a pattern h based on its frequency in H^h with Laplace smoothing having parameter ε (Manning et al. 2008).

Thus, taking as the probability model the multinomial distribution with the parameters estimated according to the ML principle, the plug-in probability of the n-th pattern in the sequence H^n is given by

$$P_{plug\text{-}in}(h^n) = \prod_{i=1}^{n} P(h^i|h^{i-1}) = \prod_{i=1}^{n} P_{\hat{\theta}_{h^{i-1}}}(h^i). \tag{6}$$

Combining together Equations 5 and 6 we obtain the following probability of the n-th pattern $h^n \in \mathcal{H}$ in the pattern sequence H^n:

$$P_{plug\text{-}in}(h^n) = \frac{\prod_{h\in\mathcal{H}} \prod_{j=0}^{usg(h)-1}(j+\varepsilon)}{\prod_{j=0}^{usg(\mathcal{H})-1}(j+\varepsilon|\mathcal{H}|)} = \frac{\prod_{h\in\mathcal{H}} \Gamma(usg(h)+\varepsilon)/\Gamma(\varepsilon)}{\Gamma(usg(\mathcal{H})+\varepsilon|\mathcal{H}|)/\Gamma(\varepsilon|\mathcal{H}|)}, \tag{7}$$

where $usg(h)$ is the number of patterns in H^n, and $usg(\mathcal{H}) = \sum_{h\in\mathcal{H}} usg(h)$ is the length of the sequence, i.e., the total number of occurrences of patterns from $\mathcal{H}$, and Γ is the gamma function. Then, the code length of h^n is

$$\begin{aligned} l(h^n) &= -\log P_{plug\text{-}in}(h^n) = \sum_{i=1}^{n} -log P_{\hat{\theta}_{H^{i-1}}}(h^i) \\ &= \log\Gamma(usg(\mathcal{H})+\varepsilon|\mathcal{H}|) - \log\Gamma(\varepsilon|\mathcal{H}|) \\ &\quad - \sum_{h\in\mathcal{H}} \left[\log\Gamma(usg(h)+\varepsilon) - \log\Gamma(\varepsilon)\right]. \end{aligned}$$

The length $l(h^n)$ can be interpreted as the sum of the log loss of the prediction errors made by sequentially predicting h^i based on the predictor in the family of multinomial distributions over $\mathcal{H}$ that would have been the best for sequentially predicting the previous patterns $h^1, \ldots, h^{i-1}$.

Appendix B The ML estimates of the parameters of the multinomial distribution

Let $\mathcal{H}$ be a set of n patterns, i.e., $n = |\mathcal{H}|$ and x_i be the number of times $h_i \in \mathcal{H}$ appears in the sequence of patterns of length N. Then, the probability of the multinomial distribution for the given sequence has the following form:

$$p(x_1, \ldots, x_n \mid p_1, \ldots, p_n) = \frac{N!}{x_1! \ldots x_n!} \prod_{i=1}^{n} p_i^{x_i}.$$

The log-likelihood function then is the following:

$$\ell(p_1, \ldots, p_n) = \log(N!) - \sum_{i=1}^{n} \log(x_i!) + \sum_{i=1}^{n} x_i \log(p_i).$$

To find the ML estimates we need to maximize this function w.r.t. $p_1, \dots, p_n$ given the constraint on the probabilities $\sum_{i=1}^{n} i = 1$. Applying the methods of Lagrange multipliers we get the following equation:

$$\mathcal{L}(p_1, \dots, p_n, \lambda) = \ell(p_1 \dots, p_n) + \lambda(1 - \sum_{i=1}^{n}(p_i)).$$

Thus

$$\begin{cases} \frac{\partial \mathcal{L}}{\partial p_i} = \frac{x_i}{p_i} - \lambda = 0, i = 1, \dots, n \\ \frac{\partial \mathcal{L}}{\partial \lambda} = 1 - \sum_{i=1}^{n} p_i = 0 \end{cases}$$

Taking into account the pseudo-counts, we get $x_i = usg(h_i) + \varepsilon$, where $usg(h_i)$ is the number of times the pattern h_i appears in the sequence.

Thus,

$$\begin{cases} p_i = \frac{usg(h_i)+\varepsilon}{\lambda}, i = 1, \dots, n \\ \sum_{i=1}^{n} p_i = 1 \end{cases}$$

Putting the last equation into the first one we get $p_i = \frac{usg(h_i)+\varepsilon}{\sum_{i=1}^{n} usg(h_i)+\varepsilon|\mathcal{H}|}$.

Appendix C MINT vs GMINT

We compare the results of MINT and GMINT w.r.t. the main characteristics, namely the number of patterns, the running time, and the compression ratio, for datasets discretized into 5, $\sqrt{|G|}$ intervals, and IPD-based discretization. The results are given in Table 5. We report values in the format $x \pm y$, where x is for GMINT, and $x \pm y$ is for MINT. For example, for "breast wisconsin", GMINT returns 54 patterns while MINT returns 5 patterns more, thus 59. The running time of GMINT is 0 seconds while the running time of MINT is 97 seconds longer. The compression ratio is 31% for both methods.

The results show that for the default settings (the number of intervals and neighbors is equal to $\sqrt{|G|}$), both algorithms return almost the same number of patterns, the compression ratio for GMINT is greater than for MINT by less than 1%. However, the execution time for middle-sized datasets is drastically smaller for GMINT.

In the other discretization settings, GMINT returns a smaller number of patterns with almost the same compression ratio and it takes much shorter time. A smaller number of patterns in GMINT can be explained by the following. In GMINT the estimate of the length gain ΔL (line 8 in Algorithm 2) is equal to the actual length gain, while in MINT we recompute the cover, and thus the actual gain can be different. Thus, when we check the condition $\Delta L > 0$ in line 8 in GMINT, the gain is exactly ΔL and then patterns h_j and h_k ensuring this gain are merged, while in MINT the total length

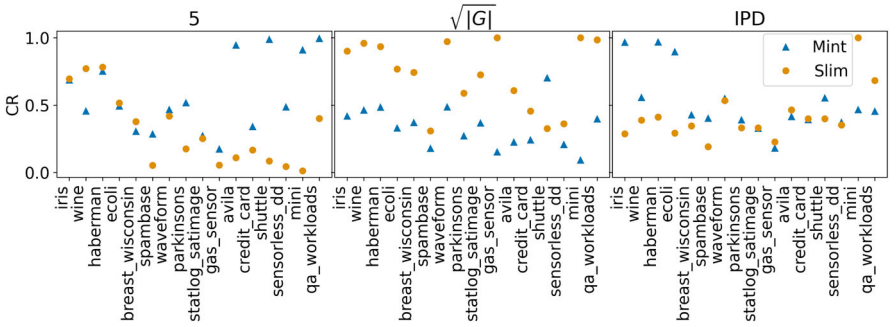


Fig. 8 Compression ratios of SLIM and MINT for different discretizations (into 5 intervals, $\sqrt{|G|}$ and by IPD discretizer), the smaller values are better

computed in line 11 can be longer, i.e., the actual gain is negative, and then patterns h_j and h_k are not merged.

Apeendix D Compression ratio of the MDL-based pattern miners

REALKRIMP mines each pattern independently from others, thus the compression ratio of a pattern set is not measurable. SLIM and MINT have different encoding schemes and compress binary and discretized datasets, respectively. Thus the compression ratios are not fully comparable. However, for the sake of completeness, we report the compression ratios that they provide in Fig. 8. As it is expected, SLIM works better in the case of a coarse discretization. For example, for equal-width discretization into 5 intervals, SLIM provides a much better compression than MINT for datasets such as "aliva", "shuttle", "sensorless dd", and "mini". However, MINT works consistently better for fine discretized datasets (Fig. 8 in the middle, where the number of intervals is $\sqrt{|G|}$). This can be explained by the fact that IPD and MINT compress the same dataset, while SLIM uses an additional binarization step and compresses a dataset containing less information, i.e., in a binarized dataset the interval order is not preserved. For IPD-discretized data, SLIM often ensures a better compression. This can be explained by the fact that IPD compresses the same data as MINT, and MINT may be unable to compress further the same dataset. By contrast SLIM compresses a dataset that was additionally binarized. Thus, the better compression of SLIM may be partially explained by artifacts caused by this data transformation.

Another important characteristic of MDL-based approaches is the *total description length* resulting from compression. Because of the different forms of data used by SLIM and MINT (binary and discrete, respectively), the algorithms may have quite different compressed total description lengths.

Nevertheless, we report them. The ratio of the total length of SLIM by the total length of MINT is given in Fig. 9. As in the case of the compression ratio, REALKRIMP does not allow computing the total length of data encoded by a pattern set, as it computes only length gains provided by single patterns. Fig. 9 shows that the total length of SLIM is usually about 2 times greater than the total length of MINT. The latter can be

Table 5 The number of selected patterns (# patt.), running time, and compression ratio (in percents). The characteristics are reported as $x \pm y$, where x is for GMINT and $x \pm y$ for MINT

	5			$\sqrt{\lvert G \rvert}$			IPD		
	# patt.	time	CR, %	# patt.	time	CR, %	# patt.	time	CR, %
iris	8+1	0+0	69−0	8+1	0+2	42−0	6+0	0+0	97+0
wine	24+0	0+3	46−1	16+0	0+3	46−0	17+1	0+1	56−2
haberman	7+2	0+0	75−0	11+0	0+5	48−0	3+0	0+0	97+0
ecoli	19+2	0+3	49−0	11+1	0+24	33−1	12−1	0+0	90−2
breast wisc.	54+5	0+97	31−0	33+2	1+122	37−0	66−5	0+79	43+0
spambase	83+2	1+265	29−0	201+6	49+57641	18−0	504+12	6+30180	41+0
waveform	1177+13	16+71594	54−0	148−2	149+118735	49−1	73+10	21+127915	55−2
parkinsons	107+11	1+532	52−0	209−6	79+195215	27−0	404+10	7+121368	39−1
statlog satim.	716-10	13+175173	31−1	260+4	207+247915	37−0	457+9	10+169612	33+0

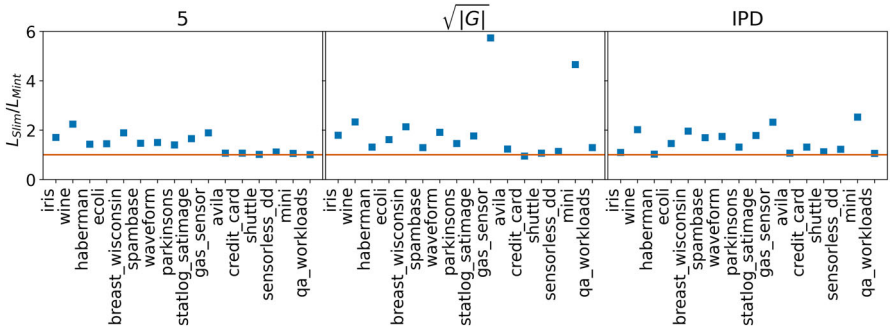


Fig. 9 The ratio of the total description length of SLIM by the total description length MINT. The values close to 1 (horizontal line) corresponds to the cases where the total description lengths provided by both methods are almost the same

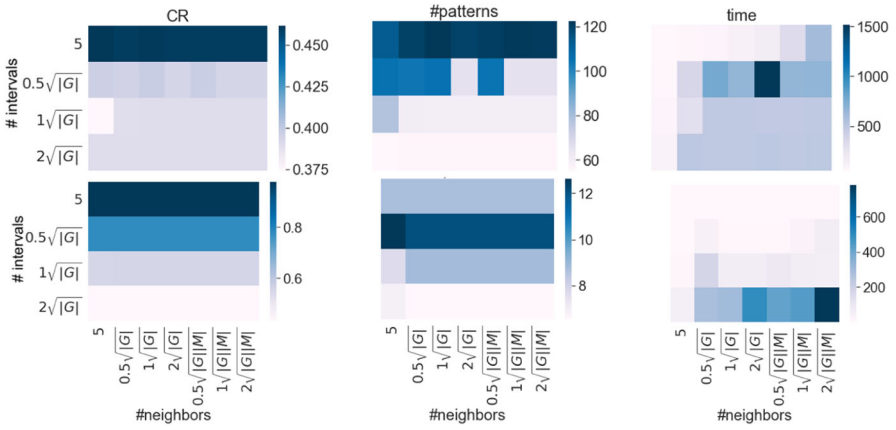


Fig. 10 Average compression ratio, number of patterns, and running time of real-world (top) and synthetic (bottom) dataset for different combinations of the parameters of MINT

explained, in particular, by the redundancy caused by data binarization. Then it can be concluded that the encoding of discretized data provided by MINT is more concise than the encoding of the binarized data used in SLIM.

Appendix E Optimal Optimal number of intervals and neighbors

In MINT we use two parameters that can affect the results: the number of discretization intervals and the number of neighbors used to compute the candidates. For a dataset with $|G|$ objects and $|M|$ attributes, we consider the following numbers: (i) $5, 0.5\sqrt{|G|}$, $\sqrt{|G|}, 2\sqrt{|G|}$ discretization intervals, and (ii) $5, 0.5\sqrt{|G|}$, $\sqrt{|G|}, 2\sqrt{|G|}, 0.5\sqrt{|G||M|}$, $\sqrt{|G||M|}, 2\sqrt{|G||M|}$ nearest neighbors.

Then we evaluate the performance of MINT regarding compression ratio, number of patterns, and running time. In Fig. 10 we show the average values for 9 real-world datasets ("iris", "waveform", "wine", "breast wisconsin", "spambase", "parkinsons",

"haberman", "ecoli", "statlog satimage") and for 30 synthetic datasets (6 types of datasets given in Fig. 5, each type has 5 versions where patterns have support 100, 200, 500, 700, and 1000).

The first figure has one dark horizontal line corresponding to discretization into 5 intervals and different numbers of neighbors. For such a coarse discretization, MINT compresses quite bad as the compression ratio is around 0.45. For finer discretizations, i.e., into $0.5\sqrt{|G|}$, $\sqrt{|G|}$, and $2\sqrt{|G|}$ in the next 3 light-blue horizontal lines, the compression ratio is around 0.40 and does not changes a lot when the number of candidates changes (i.e., the colors on the horizontal line do not change a lot).

For finer discretized datasets the number of neighbors does not affect a lot the results. It appears that with a large number of discretized intervals, the results of MINT are less affected by the chosen heuristics.

Taking into account the considered three characteristics, the best and the most "stable" results in the shortest time are achieved by splitting the attribute range into $\sqrt{|G|}$ intervals and by taking $\sqrt{|G|}$ nearest neighbors to compute pattern candidates.

References

Akoglu L, Tong H, Vreeken J, Faloutsos C (2012) Fast and reliable anomaly detection in categorical data. In: Proceedings of the 21st ACM international conference on information and knowledge management. ACM, pp 415–424

Bariatti F, Cellier P, Ferré S (2020) GraphMDL: graph pattern selection based on minimum description length. In: International symposium on intelligent data analysis (IDA). Springer, pp 54–66

Bondu A, Boullé M, Lemaire V (2010) A non-parametric semi-supervised discretization method. Knowl Inf Syst 24(1):35–57

Boullé M (2006) MODL: a Bayes optimal discretization method for continuous attributes. Mach Learn 65(1):131–165

Budhathoki K, Vreeken J (2015) The difference and the norm—characterising similarities and differences between databases. In: Joint European conference on machine learning and knowledge discovery in databases. Springer, pp 206–223

Calders T, Goethals B, Jaroszewicz S (2006) Mining rank-correlated sets of numerical attributes. In: Proceedings of the 12th ACM SIGKDD international conference on knowledge discovery and data mining, pp 96–105

Coenen F (2003) The LUCS-KDD discretised/normalised ARM and CARM data library. Department of CS, The University of Liverpool, UK http://www.csc.liv.ac.uk/~frans/KDD/Software/LUCS_KDD_DN

Dash R, Lochan PR, Rasmita D (2011) Comparative analysis of supervised and unsupervised discretization techniques. Int J Adv Sci Technol 2(3):29–37

Dua D, Graff C (2017) UCI machine learning repository. http://archive.ics.uci.edu/ml

Faas M, van Leeuwen M (2020) Vouw: geometric pattern mining using the MDL principle. In: International symposium on intelligent data analysis (IDA). Springer, pp 158–170

Fayyad UM, Irani KB (1993) Multi-interval discretization of continuous-valued attributes for classification learning. In: Ruzena B (ed) Proceedings of the 13th international joint conference on artificial intelligence. Morgan Kaufmann, pp 1022–1029

Galbrun E (2020) The minimum description length principle for pattern mining: a survey. arXiv:2007.14009

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

Jain AK (2010) Data clustering: 50 years beyond K-means. Pattern Recogn Lett 31(8):651–666

Jeantet I, Miklós Z, Gross-Amblard D (2020) Overlapping hierarchical clustering (OHC). In: Proceedings of the 18th international symposium on intelligent data analysis (IDA), volume 12080 of lecture notes in computer science, vol 12080. Springer, pp 261–273

Kang Y, Wang S, Liu X, Lai H, Wang H, Miao B (2006) An ICA-based multivariate discretization algorithm. In: International conference on knowledge science, engineering and management. Springer, pp 556–562

Kaytoue M, Kuznetsov SO, Napoli A (2011) Revisiting numerical pattern mining with formal concept analysis. In: Twenty-second international joint conference on artificial intelligence

Kontkanen P, Myllymäki P (2007) MDL histogram density estimation. In: Artificial intelligence and statistics, pp 219–226

Makhalova T, Trnecka M (2021) From-below Boolean matrix factorization algorithm based on MDL. Adv Data Anal Classif 15(1):37–56

Makhalova T, Kuznetsov SO, Napoli A (2019) Numerical pattern mining through compression. In: 2019 data compression conference (DCC). IEEE, pp 112–121

Manning CD, Raghavan P, Schütze H (2008) Introduction to Information Retrieva. Cambridge University Press, Cambridge

Mehta S, Parthasarathy S, Yang H (2005) Toward unsupervised correlation preserving discretization. IEEE Trans Knowl Data Eng 17(9):1174–1185

Miettinen P, Vreeken J (2014) MDL4BMF: minimum description length for Boolean matrix factorization. ACM Trans Knowl Discov Data: TKDD 8(4):1–31

Nguyen H-V, Müller E, Vreeken J, Böhm K (2014) Unsupervised interaction-preserving discretization of multivariate data. Data Min Knowl Disc 28(5–6):1366–1397

Proença HM, van Leeuwen M (2020) Interpretable multiclass classification by MDL-based rule lists. Inf Sci 512:1372–1393

Rissanen J (1983) A universal prior for integers and estimation by minimum description length. Ann Stat 11(2):416–431

Rissanen J, Speed TP, Bin Yu (1992) Density estimation by stochastic complexity. IEEE Trans Inf Theory 38(2):315–323

Siebes A, Vreeken J, van Leeuwen M (2006) Item sets that compress. In: Proceedings of the 2006 SIAM international conference on data mining. SIAM, pp 395–406

Smets K, Vreeken J (2012) Slim: directly mining descriptive patterns. In: Proceedings of SIAM. SIAM, pp 236–247

Srikant R, Agrawal R (1996) Mining quantitative association rules in large relational tables. In: Proceedings of the ACM SIGMOD international conference on management of data, pp 1–12

Tatti N (2013) Itemsets for real-valued datasets. In: 2013 IEEE 13th international conference on data mining. IEEE, pp 717–726

Tatti N, Vreeken J (2008) Finding good itemsets by packing data. In: Eighth IEEE international conference on data mining. IEEE, pp 588–597

Tatti N, Vreeken J (2012a) The long and the short of it: summarising event sequences with serial episodes. In: Proceedings of the 18th ACM SIGKDD international conference on knowledge discovery and data mining, pp 462–470

Tatti N, Vreeken J (2012b) Discovering descriptive tile trees—by mining optimal geometric subtiles. In: Proceedings of the European conference on machine learning and knowledge discovery in databases (ECML-PKDD), lecture notes in computer science, vol 7523. Springer, pp 9–24

van Craenendonck T, Dumancic S, Blockeel H (2017) COBRA: a fast and simple method for active clustering with pairwise constraints. In: Proceedings of the 26 international joint conference on artificial intelligence (IJCAI), pp 2871–2877

Vreeken J, Tatti N (2014) Interesting patterns. In: Aggarwal CC, Han J (eds) Frequent pattern mining. Springer, Berlin, pp 105–134

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

Witteveen J (2012) Mining hyperintervals—getting to grips with real-valued data. Bachelor's thesis

Witteveen J, Duivesteijn W, Knobbe A, Grünwald P (2014) Realkrimp—finding hyperintervals that compress with MDL for real-valued data. In: International symposium on intelligent data analysis. Springer, pp 368–379