Pattern Mining in Numerical Data: Extracting Closed Patterns and their Generators

Mehdi Kaytoue, Sergei O. Kuznetsov, Amedeo Napoli

Pattern Mining in Numerical Data: Extracting Closed Patterns and their Generators

Mehdi Kaytoue — Sergei O. Kuznetsov — Amedeo Napoli

N° 7416

September 2010

Knowledge and Data Representation and Management

ISSN 0249-6399 ISRN INRIA/RR--7416--FR+ENG

Pattern Mining in Numerical Data: Extracting Closed Patterns and their Generators

Mehdi Kaytoue, Sergei O. Kuznetsov, Amedeo Napoli

Theme : Knowledge and Data Representation and Management
Équipe-Projet Orpailleur

Rapport de recherche n° 7416 — September 2010 — 25 pages

Abstract: In this paper we study the extraction of closed patterns associated to their generators in numerical data. Many works have addressed the problem of extracting itemsets for generating association rules. Considering numerical data, an appropriate discretization is most of the time necessary, in order to split attribute ranges into intervals maximizing some interest functions, e.g. support, confidence, or other statistical measures. We investigate here an alternative point of view using pattern structures in Formal Concept Analysis. Pattern structures can be efficiently used to extract closed patterns without any prior discretization. Two original and efficient algorithms for characterizing frequent closed patterns and their generators in numerical data are proposed and experimented. Finally, we conclude showing the usefulness of such patterns in classification problems and privacy preserving data-mining.

Key-words: Formal concept analysis, pattern mining, numerical data

Sergei O. Kuznetsov is a Professor at the State University Higher School of Economics (HSE) – Kirpichnaya 33/5 – 125219 Moscow, Russia

Extraction de motifs fermés et gnrateurs à partir de données numériques

Résumé : Dans cet article, nous tudions l'extraction de motifs ferms et leurs gnrateurs partir de donnes numriques. De nombreux travaux s'intressent l'extraction de motifs pour la gnration de rgles d'association dans le cadre de la dcouverte de connaissances. Concernant les donnes numriques, une tape de discrtisation est gnralement ncessaire, afin de dcouper les domaines des attributs en un certain nombre d'intervalles, maximisant certaines fonctions d'intrt, comme le support ou la confiance. Nous proposons ici une mthode alternative se basant sur la notion de structures de patrons dfinies dans le cadre de l'analyse formelle de concepts (FCA). Les structures de patrons peuvent tre efficacement utilises pour extraire des motifs ferms partir de donnes numriques sans discrtisation pralable des donnes. Nous proposons alors deux algorithme originaux et efficaces pour caractriser et extraire les motifs ferms et gnrateurs dans les donnes numriques. Nous concluons sur l'utilit de tels motifs pour des tches de classification, mais aussi d'anonymisation de donnes.

Mots-clés : Extraction de motifs, donnes numeriques, analyse formelle de concepts

1 Introduction

We discuss the mining of numerical data with symbolic methods based on an adaptation of classical pattern mining methods. Numerical data are commonly found in many application domains, and especially in agronomy, biology, chemistry, or medicine. Usually, such data are analyzed by methods based either on data analysis, statistics, or probabilities [13], e.g. clustering.These methods are usually efficient but also return results hard to understand and to interpret. Indeed, when the emphasis is put on knowledge discovery, it is expected that the results returned by a data mining process can be interpreted in terms of knowledge units [19]. This is one reason for combining symbolic and numerical methods.

In parallel, the application of pattern mining methods is well studied on binary data as well as the design of concept lattices using Formal Concept Analysis (FCA [9]). For example, applying FCA and extensions such as Relational Concept Analysis [22] in text mining can return units that can be embedded in an ontology [4]. Accordingly, it seems interesting to apply pattern mining techniques on numerical data for extracting patterns, for data analysis and ontology engineering purposes.

The present work is rooted both in FCA and pattern mining with the objective of extracting interval patterns from numerical data. Our approach is based on “pattern structures” where complex descriptions can be associated with objects [8]. Such descriptions can be numbers, intervals, and even graphs [16]. In [15], in the context of gene expression data mining, we introduced pattern structures for numerical data, and showed how to extract closed interval patterns. Intuitively, an interval pattern is a vector of intervals, each dimension corresponding to a range of values of a given attribute. In the present paper, we complete and extend this first attempt. Considering numerical data, some general characteristics of equivalence classes remain, e.g. one maximal element which is a closed pattern and possibly several generators which are minimal patterns w.r.t. a subsumption relation defined on patterns. We also provide a semantic to interval patterns in the Euclidean space, and design and experiment algorithms to extract frequent closed interval patterns and their generators.

The problem of mining patterns in numerical data is usually referred as quantitative itemset/association rule mining [25]. Generally, an appropriate discretization splits attribute ranges into intervals maximizing some interest functions, e.g. support, confidence. However, none of these works discusses the notion of equivalence classes, closed patterns, and generators, and this is one of the originality of the present paper.

The plan of the paper is as follows. After the introduction, Section 2 explains the problem of extracting pattern from numerical data. Then, closed interval patterns, generators, and equivalence classes are properly defined in Section 3, and a semantic is provided. Section 4 discusses links with classical itemset mining in binary data. Section 5 details different algorithms for extracting frequent closed patterns and their generators in numerical data. Experiments on the proposed algorithms and a discussion terminate the paper, showing the usefulness of such patterns in classification problems and privacy preserving data-mining.

2 Problem Specification

We propose a definition of interval patterns for numerical data following ideas in [8, 15]. Intuitively, each object of a numerical dataset is a vector of numbers, where each dimension corresponds to an attribute. Accordingly, an interval pattern is a vector of intervals, where each dimension describes the range of possible values for a given numerical attributes associated with some objects. We only consider finite intervals.

Definition 2.1 (Numerical dataset) *A numerical dataset is given by a set of objects G, a set of numerical attributes M, each attribute $m \in M$ having for range a set of real numbers W_m. We denote by $m(g) = w$ the fact that w is the value of attribute m for object g.*

	m_1	m_2	m_3
g_1	5	7	6
g_2	6	8	4
g_3	4	8	5
g_4	4	9	8
g_5	5	8	5

Table 1: A numerical dataset.

Definition 2.2 (Interval pattern and support) *In a numerical dataset, an interval pattern is a vector of intervals $d = \langle [a_i, b_i] \rangle_{i \in \{1,...,|M|\}}$ where $a_i, b_i \in W_{m_i}$, and each component corresponds to an attribute following a canonical order on vector dimensions, and $|M|$ denotes the number of attributes. An object g is in the image of an interval pattern $\langle [a_i, b_i] \rangle_{i \in \{1,...,|M|\}}$ when $m_i(g) \in [a_i, b_i]$, $\forall i \in \{1, ..., |M|\}$. The support $sup(d)$ of d is the cardinality of the image of d.*

Running example. Table 1 is a numerical dataset with objects in $G = \{g_1, ..., g_5\}$, attributes in $M = \{m_1, m_2, m_3\}$. The range of m_1 is $W_{m_1} = \{4, 5, 6\}$, and we have $m_1(g_1) = 5$. Here, we do not consider either missing values or multiple values for an attribute. $\langle [5, 6], [7, 8], [4, 6] \rangle$ is an interval pattern in Table 1, where a vector dimension i corresponds to an attribute m_i. Its image is $\{g_1, g_2, g_5\}$ and its support is 3.

Definition 2.3 (Interval pattern search space) *Given a set of attributes $M = \{m_i\}_{i \in \{1,|M|\}}$, the search space of interval patterns is the set D of all interval vectors $\langle [a_i, b_i] \rangle_{i \in \{1,...,|M|\}}$, with $a_i, b_i \in W_{m_i}$ and $a_i \leq b_i$. The size of the search space is given by*

$$|D| = \prod_{i \in \{1,...,|M|\}} \frac{|W_{m_i}| \times (|W_{m_i}| + 1)}{2}$$

where $\frac{|W_{m_i}| \times (|W_{m_i}|+1)}{2}$ is the number of possible intervals for the attribute m_i.

For example, all possible intervals for m_1 are in $\{[4, 4], [5, 5], [6, 6], [4, 5], [5, 6], [4, 6]\}$. Considering also attributes m_2 and m_3, the interval pattern search space is naturally larger, composed of $6 \times 6 \times 10 = 360$ interval patterns in our example.

Among well-known solutions to deal with "pattern flooding" in data-mining, one is to efficiently mine frequent patterns, i.e. patterns having support greater than a given threshold, while a second is to define condensed representations of patterns [24], e.g. closed patterns, (minimal) generators (also called key-sets, free-sets), etc. While generators can be preferable to closed patterns following the minimum descriptions length principle [20], closed patterns and their generators are known to be crucial for extracting valid and interesting association rules [3]. Therefore, we discuss and solve the following problems.

Problem 1: *Mining frequent closed interval patterns.* Whereas an algorithm was proposed for mining closed interval patterns in [15], it addressed the dual problem of un-frequent interval patterns mining, i.e. with support smaller than given threshold. We propose the algorithm *MinIntChange* for efficiently mining frequent closed interval patterns. Most importantly, this algorithm is useful for considering the two next problems.

Problem 2: *Mining interval pattern generators.* Closed patterns determine equivalence classes. One should expect that these classes have minimal elements w.r.t. a subsumption relation on patterns, called *interval pattern generators.* We propose to characterize these notions and to design an algorithm to efficiently mine frequent generators, called *MinIntChangeG.*

Problem 3: *Associating generators to their closure. MinIntChangeG* can provide each generator with its closure, allowing to produce valid and confident association rules.

Problem 4: *Mining equivalent binary data.* In [15], we showed that numerical data can be turned into binary with a so-called interordinal scaling, and that resulting binary data (i) can be mined with existing itemset mining algorithms, and (ii) there is a one-to-one correspondence between closed interval patterns and closed itemsets. However, we showed that closed interval patterns have better representation, avoid a local redundancy, and are much more efficient to mine directly in numerical data. Therefore, we should ensure that the same holds for generators, and than our algorithms are more efficient that classical algorithms in these particular binary data.

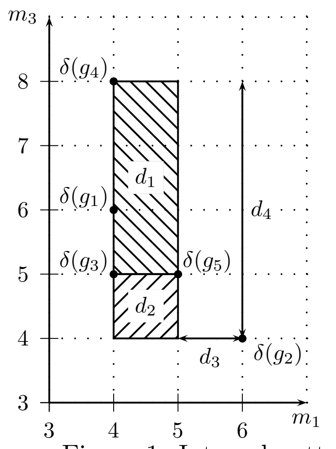
Before solving these problems, we properly define (freqeunt)(closed) interval patterns (and generators) and their semantics in $\mathbb{R}^{|\mathbb{M}|}$.

3 Interval Patterns: Semantics and Definitions

3.1 Semantics

Consider a numerical dataset with objects in G and numerical attributes in M. An interval pattern d is a $|M|$-dimensional vector of intervals, and can be represented by a hyperrectangle (or rectangle for short) in Euclidean space $\mathbb{R}^{|M|}$, whose sides are parallel to the coordinate axes. This geometrical representation will be considered as the semantics of interval patterns. Formally, an interpretation is given by $\mathcal{I} = (\mathbb{R}^{|M|}, (.)^{\mathcal{I}})$ with $\mathbb{R}^{|M|}$ the interpretation domain, and $(.)^{\mathcal{I}} : D \to \mathbb{R}^{|M|}$ the interpretation function.

Example. When illustrating patterns in $\mathbb{R}^{|\mathbb{M}|}$, we consider the numerical dataset of Table 1 with attributes m_1 and m_3 only (it is more convenient here to work on two dimensions). The Figure 1 (left) gives four interval patterns d_1, d_2, d_3, d_4 and their representation in $\mathbb{R}^2$. In two dimensions, a pattern with two intervals with same left and right borders is a point, while a pattern having only one interval with same borders is a segment, e.g. d_3 and d_4. Otherwise, a pattern is represented by a rectangle, e.g. d_1 and d_2.



$d_1 = \langle[4,5],[5,8]\rangle$
$d_1^\square = \{g_1, g_3, g_4, g_5\}$
$d_2 = \langle[4,5],[4,5]\rangle$
$d_2^\square = \{g_3, g_5\}$
$d_3 = \langle[5,6],[4,4]\rangle$
$d_3^\square = \{g_2\}$
$d_4 = \langle[6,6],[4,8]\rangle$
$d_4^\square = \{g_2\}$

Figure 1: Interval patterns in the Euclidean space.

3.2 Ordering interval patterns

A basic idea in pattern mining is to define an intersection on patterns allowing to build more general patterns, i.e. shared by more objects. As stated in [8], the set-theoric intersection has the properties of an infimum $\sqcap$ in a semi-lattice $(D, \sqcap)$, i.e. idempotent, commutative, and associative. Accordingly, we introduced an infimum operation on interval patterns [15]:

Definition 3.1 (Infimum of Interval patterns) *Given two intervals patterns* $c = \langle[a_i, b_i]\rangle_{i\in\{1,\ldots,|M|\}}$, *and* $d = \langle[e_i, f_i]\rangle_{i\in\{1,\ldots,|M|\}}$, *their infimum is given by* $c \sqcap d = \langle[min(a_i, e_i), max(b_i, f_i)]\rangle_{i\in\{1,\ldots,|M|\}}$.

The infimum of several patterns is interpreted as the convex hull of their hyperrectangles in $\mathbb{R}^{|\mathbb{M}|}$, e.g. $d_1 \sqcap d_2 = \langle[4,5],[4,8]\rangle$ in Figure 1. This definition induces partial order, or subsomption relation $\sqsubseteq$ on interval patterns, knowing that $c \sqcap d = c \Leftrightarrow c \sqsubseteq d$.

Definition 3.2 (Subsumption relation) *Given two interval patterns c and d,* $c \sqsubseteq d$ *holds if* $d^{\mathcal{I}} \subseteq c^{\mathcal{I}}$.

This means that two interval patterns c and d are comparable whenever $c^{\mathcal{I}} \subseteq d^{\mathcal{I}}$ or $d^{\mathcal{I}} \subseteq c^{\mathcal{I}}$ and that patterns with "larger" intervals are subsumed by patterns with "smaller" intervals. For example, $\langle[4,5],[4,8]\rangle \sqsubseteq \langle[4,5],[4,5]\rangle$ but $\langle[4,5],[4,5]\rangle$ and $\langle[4,5],[5,8]\rangle$ are not comparable.

Example. We consider in this example one-dimensional interval patterns. Choosing attribute m_1 from Table 1, the set of all possible interval patterns is $D_{m_1} =$

$\{[4,4],[5,5],[6,6],[4,5],[5,6],[4,6]\}$. The semi-lattice $(D,\sqcap)$, or equivalently $(D,\sqsubseteq)$ is given in Figure 2. The interval labelling a node is the infimum of all intervals labelling its descending nodes, e.g. $[4,5] = [4,4] \sqcap [5,5]$, and is also subsumed by these intervals, e.g. $[4,5] \sqsubseteq [5,5]$ and $[4,5] \sqsubseteq [4,4]$.

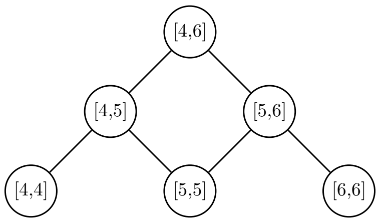


Figure 2: Diagram of $(D_{m_1},\sqcap)$ or equivalently$(D_{m_1},\sqsubseteq)$.

Finally, the support of an interval pattern d is interpreted as the the number of objects described by a rectangle included in $d^{\mathcal{I}}$, e.g. support of d_1 is four in Figure 1, with $\delta(g)$ represents the rectangle describing object $g \in G$.

3.3 Pattern Structures in FCA

The following definitions formally define pattern structures, involving a closure operator on patterns, based on a Galois connection. Pattern structure is an extension of well-know *formal contexts* (binary tables) to complex data in FCA [9, 8].

Definition 3.3 (Pattern structure) *Let G be a set of objects, let $(D,\sqcap)$ be a meet-semi-lattice of object descriptions, called patterns, and let $\delta : G \longrightarrow D$ be a mapping: $(G,(D,\sqcap),\delta)$ is called a pattern structure.*

Definition 3.4 *Let the two following operators $(.)^{\square}$ defined as follows.*

$$A^{\square} = \bigsqcap_{g \in A} \delta(g), \quad for\ A \subseteq G$$
$$d^{\square} = \{g \in G | d \sqsubseteq \delta(g)\}, \quad for\ d \in (D,\sqcap).$$

These operators form a Galois connection between $(\mathfrak{P}(G),\subseteq)$ and $(D,\sqsubseteq)$. The operator $(.)^{\square\square}$ is a closure operator.

Example. Considering the example of Table 1. $(D,\sqsubseteq)$ is the finite ordered set of all interval patterns. $\delta(g) \in D$ is the pattern associated to an object $g \in G$. Then:

$$\begin{aligned}
\langle [5,6],[7,8],[4,8] \rangle^{\square} &= \{g \in G | \langle [5,6],[7,8],[4,8] \rangle \sqsubseteq \delta(g)\} \\
&= \{g_1, g_2, g_5\} \\
\{g_1, g_2, g_5\}^{\square} &= \delta(g_1) \sqcap \delta(g_2) \sqcap \delta(g_3) \\
&= \langle [5,6],[7,8],[4,6] \rangle
\end{aligned}$$

This means that $\langle [5,6],[7,8],[4,8] \rangle$ is not a closed interval pattern, its closure being $\langle [5,6],[7,8],[4,6] \rangle$. The first operator applies to an arbitrary description $d \in (D,\sqcap)$ and returns the set of objects described by rectangles included in $d^{\mathcal{I}}$. Dually, the second operator applies to a of objects $A \subseteq G$ and returns the convex hull of their interpretation, i.e. a rectangle.

Based on these definitions, we now define the notions of (frequent) closed interval pattern ((F)CIP), equivalence classes of patterns and (frequent) interval patterns generators ((F)IPG), adapted from the classical binary case [21]. We illustrate these definitions with two dimensional interval patterns, and their representation in Figure 1, i.e. considering attributes m_1 and m_3 only.

Definition 3.5 (Equivalence class) *Let $image(d)$ be the function that assigns to each interval pattern the set of objects supporting d, i.e. $image(d) = d^\square$. Two interval patterns c and d are said equivalent iff they have the same image and we write $c \cong d$. The set of patterns that are equivalent to a pattern d is denoted by $[d] = \{c|c \cong d\}$ and is called the equivalence class of d.*

Example. $\langle[4,5],[6,8]\rangle \cong \langle[4,6],[6,8]\rangle$ as they have the same image $\{g_1, g_4\}$.

Definition 3.6 (Closed interval pattern) *A pattern d is closed if there does not exist any pattern e such as $d \sqsubseteq e$ with $d \cong e$.*

Example. $\langle[4,6],[6,8]\rangle$ is not closed as $\langle[4,6],[6,8]\rangle \sqsubseteq \langle[4,5],[6,8]\rangle$, these two patterns having same image, i.e. $\{g_1, g_3, g_4, g_5\}$. $\langle[4,6],[6,8]\rangle$ is closed.

Definition 3.7 (Interval pattern generator) *A pattern d is a generator if there does not exist a pattern e such as $e \sqsubseteq d$ with $d \cong e$.*

Example. $\langle[4,6],[5,8]\rangle$ and $\langle[4,5],[4,8]\rangle$ are the generators of the closed interval pattern $d_1 = \langle[4,5],[5,8]\rangle$ with image $\{g_1, g_3, g_4, g_5\}$.

Definition 3.8 (Frequent Interval pattern) *A pattern d is frequent if its image has a higher cardinality than a given minimal support threshold minSup, i.e $|d^\square| \geq minSup$. Otherwise, d is not frequent.*

Example. Among the four patterns in Figure 1, d_1 is the only frequent interval pattern with $minSup = 3$.

An equivalence class is a set of interval patterns having the same image. According to the defined closure operator, each class is provided with a unique CIP. The interpretation of this closed pattern is the rectangle with smallest area, while generators are rectangles with largest area.

We dedicate a particular attention to interval patterns with null support. In Figure 1, such patterns correspond to rectangles, segments or points containing no object description from the dataset, e.g. $c_1 = \langle[6,6],[5,8]\rangle$, $c_2 = \langle[5,6],[6,8]\rangle$, $c_3 = \langle[4,4],[4,4]\rangle$. Such patterns would not exist if each point in the rectangle $\langle[4,6],[4,8]\rangle$ were covered by some object of the dataset (since the search space is finite). If interval patterns with null support exist, their equivalence class should have a closed element with one or more generators. However, the closed pattern of null support does not exist, since it should subsume any closed pattern of support 1. Any CIP with support 1 is defined by $g^\square$ for some $g \in G$. Since dealing with numerical attributes with domains values in $\mathbb{R}$, intervals of $g^\square$ are degenerate (same left and right borders), e.g. $\delta(g1) = \langle[5,5],[7,7],[6,6]\rangle$. Therefore, we cannot find a subsumer of this pattern: it is not defined (any degenerate interval has no subintervals). When existing, the generators of null support provide a meaningful information: it characterizes the largest subspaces of the data covered by no objects.

$m_3 \geq 8$				×	
$m_3 \geq 6$	×			×	
$m_3 \geq 5$	×		×	×	×
$m_3 \geq 4$	×	×	×	×	×
$m_3 \leq 8$	×	×	×	×	×
$m_3 \leq 6$	×	×	×		×
$m_3 \leq 5$		×	×		×
$m_3 \leq 4$		×			
$m_2 \geq 9$				×	
$m_2 \geq 8$		×	×	×	×
$m_2 \geq 7$	×	×	×	×	×
$m_2 \leq 9$	×	×	×	×	×
$m_2 \leq 8$	×	×	×		×
$m_2 \leq 7$	×				
$m_1 \geq 6$		×			
$m_1 \geq 5$	×	×			×
$m_1 \geq 4$	×	×	×	×	×
$m_1 \leq 6$	×	×	×	×	×
$m_1 \leq 5$	×		×	×	×
$m_1 \leq 4$			×	×	
	g_1	g_2	g_3	g_4	g_5

Table 2: Interordinally scaled context encoding the dataset from Table 1.

4 Interval Patterns in Binary Data

In this section, we recall how numerical data can be turned into binary with a so-called interordinal scaling. This data transformation is defined in the framework of formal concept analysis (FCA) [9], and allows to produce binary data from which interval patterns can be extracted [15]. For making the paper self-contained, we recall the basics of FCA necessary for understanding the following. Most importantly, we show that, in these particular binary data, collections of closed itemsets and generators highlight two forms of redundancy, leading to design efficient algorithms working directly on numerical data in the next section.

4.1 Formal Concept Analysis

FCA starts with a formal context (G, N, I) where G denotes a set of objects, N a set of attributes, or items, and $I \subseteq G \times N$ a binary relation between G and N. The statement $(g, n) \in I$ is interpreted as "the object g has attribute n". The two operators $(\cdot)'$ define a Galois connection between the powersets $(\mathfrak{P}(G), \subseteq)$ and $(\mathfrak{P}(N), \subseteq)$, with $A \subseteq G$ and $B \subseteq N$:

$$A' = \{n \in N \mid \forall g \in A : gIn\}$$
$$B' = \{g \in G \mid \forall n \in B : gIn\}$$

For $A \subseteq G$, $B \subseteq N$, a pair (A, B), such that $A' = B$ and $B' = A$, is called a *(formal) concept*. In (A, B), the set A is called the *extent* and the set B the *intent* of the concept (A, B). Concepts are partially ordered by $(A_1, B_1) \leq (A_2, B_2) \Leftrightarrow A_1 \subseteq A_2$ ($\Leftrightarrow B_2 \subseteq B_1$). With respect to this partial order, the set of all formal concepts forms a complete lattice called the *concept lattice* of (G, N, I).

From an itemset-mining point of view, concept intents correspond to closed itemsets, since $(.)''$ is a closure operator. Moreover, a subset $B \subseteq N$ is a generator iff $\nexists C \subset B$ with $C' = B'$. An equivalence class is a set of itemsets with same closure. Therefore, closed itemsets can be computed either with FCA algorithms (compared in [18]), or algorithms of the data-mining community, e.g. Charm [27], FP-Growth [12].

4.2 Interordinal Scaling

Conceptual scaling is often used for discretizing numerical data and obtaining a (binary) formal context [9]. Given a numerical attribute, the search space of interval patterns can be expressed in terms of binary attributes, or items, thanks to *interordinal scaling*. We recall here a basic definition while more details lie in [9, 15, 17].

In FCA, a numerical dataset is described by a many-valued context (G, M, W, J) where G is a set of objects, M a set of numerical attributes, W a set of real numbers, and J a ternary relation defined on the Cartesian product $G \times M \times W$. $(g, m, w) \in J$ or simply $m(g) = w$ means that the object g takes the value w for the attribute m.

Definition 4.1 (Interordinal scaling) *Given a numerical attribute m with value domain the set W_m of real numbers, interordinal scaling builds $2 \times |W_m|$ binary attributes, denoted by "$m \leq w$" and "$m \geq w$", $\forall w \in W_m$, called "interordinal scale attributes" or IS-items for short.*

Definition 4.2 (Interordinal scaled context) *A formal context (G, N, I) is an interordinal scaled context when it results from the application of interordinal scaling to numerical context (G, M, W, J). N is the set of all IS-items of the form "$m \leq w$" or "$m \geq w$" for each numerical attribute $m \in M$ and value $w \in W_m$. An object g has an IS-item "$m \leq w$" (resp. "$m \geq w$") iff $m(g) \leq w$ (resp. $m(g) \geq w$).*

Example. Table 2 gives the tabular representation of the interordinally scaled formal context built from Table 1. Object g_1 owns the IS-item $m_1 \leq 5$ (denoted by a cross $\times$) but not $m_1 \leq 4$ since $m_1(g_1) = 5$.

4.3 Interval Patterns and IS-Itemsets

It is possible to apply classical mining algorithms to process the binary table for extracting itemsets composed of IS-items. These itemsets are called IS-itemsets in the following, and are linked with interval patterns as follows [15].

An IS-itemset as an interval pattern. An IS-itemset P is composed of IS-items of the forms $m_i \leq w$ and $m_i \geq w$ for some $w \in W_{m_i}$. It is represented by the interval pattern $d = \langle [a_i, b_i] \rangle_{i \in \{1,\dots,|M|\}}$, where

- a_i is the maximum of the values w in IS-items $m_i \geq w$, and $min(W_{m_i})$ if $m_i \geq w \notin P$.
- b_i is the minimum of the values w in IS-items $m_i \leq w$, and $max(W_{m_i})$ if $m_i \leq w \notin P$.

For example, $\{m_1 \leq 5, m_1 \leq 6, m_1 \geq 4, m_2 \leq 9, m_2 \geq 7\}$ corresponds to $\langle [4,5], [7,9], [4,8] \rangle$, i.e. the smallest interval pattern w.r.t. $\sqsubseteq$ with same image.

An interval pattern as an IS-itemset. Let $d = \langle [a_i, b_i] \rangle_{i \in \{1,\dots,|M|\}}$ be an interval pattern. An IS-itemset representing d is a set of IS-attributes, $\forall i \in [1, |M|]$.

- $m_i \leq b_i$ if $a_i = min(W_{m_i})$
- $m_i \geq a_i$ if $b_i = max(W_{m_i})$
- $m_i \geq a_i$ and $m_i \leq b_i$ otherwise.

For example, the IS-itemset corresponding to $\langle [4,5][7,9][4,8] \rangle$ is $\{m_1 \leq 5\}$, i.e. the smallest set of IS-items with same image.

We detail in the following some problem when mining IS-itemsets. First, we show that closed IS-itemsets involve a local redundancy making them hard to mine. Secondly, we show that IS-itemsets generators do not behave in the same way, but involve another kind of redundancy that alter their mining.

Local redundancy of IS-itemsets. Extracting all IS-itemsets in our example returns $31,487$ IS-itemsets. This is surprising since there are only 360 possible interval patterns. In fact, a lot of IS-itemsets are locally redundant. For example, $\{m_1 \leq 5\}$ and $\{m_1 \leq 5, m_1 \leq 6\}$ both correspond to the interval pattern $\langle [4,5], [7,9], [4,8] \rangle$. Indeed, the constraint $m_1 \leq 6$ is weaker than $m_1 \leq 5$ on the set of values W_{m_1}.

Definition 4.3 *Given two IS-items $n_1, n_2 \in N$, with same sign $\leq$ or $\geq$ and numerical attribute, n_1 characterizes a weaker constraint than n_2 if $n_2' \subseteq n_1'$. n_1 is a redundant condition with respect to n_2.*

Proposition 4.1 *An arbitrary IS-itemset $N_1 \subseteq N$ is locally redundant iff it contains two IS-items such as one is a redundant condition with respect to the other one.*

Example. $\{m_1 \leq 5, m_1 \leq 6\}$ and $\{m_1 \leq 4, m_1 \leq 5, m_1 \leq 6\}$ are both locally redundant while $\{m_1 \leq 5\}$ and $\{m_1 \leq 5, m_3 \geq 5\}$ are not. Intuitively, in $\{m_1 \leq 5, m_1 \leq 6\}$ the item $m_1 \leq 6$ brings no new information on the description of the itemset image.

Proposition 4.2 *Except G', any closed IS-itemset $P \subseteq N$ is locally redundant and $|P| > 2|M|$.*

Proof. By definition of interordinal scaling, we have $G' = \{m_i \leq max(W_{m_i}), m_i \geq min(W_{m_i})\}_{\forall m_i \in M}$, hence $|G'| = 2|M|$. Any other closed itemset P is such that $G' \subset P$: it is locally redundant.

Proposition 4.3 *If $P \subseteq N$ is an IS-itemset generator, then $|P| \leq 2|M|$, and P is not locally redundant.*

Proof. Suppose that P is a generator with $|P| > 2|M|$. Since IS-items are of the form, either "$m \leq w$" or "$m \geq w$" for $m \in M$ and $w \in W_m$, P contains at least two itemsets of one of these form. Therefore, one characterizes a redundant condition and removing it from P does not change its image, leading to a contradiction. Moreover, if P_1 is redundant, $P_1 \subset P_2$ implies that P_2 is also redundant.

Global redundancy of IS-itemsets generators. Due to local redundancy, we showed in [15] that closed IS-itemsets are hard to mine with classical closed itemset mining algorithms. It seems that IS-itemset generators have a good property to be mined, since not affected by local redundancy. But we remark here another kind of redundancy, called global redundancy: it happens that two different and incomparable IS-itemsets generators correspond to two different interval pattern generators, but one subsuming the other, i.e. one is not an interval pattern generator according to the semantic in $\mathbb{R}$. For example, taking the binary table 2, both IS-itemsets $N_1 = \{m_1 \leq 4, m_3 \leq 5\}$ and $N_2 = \{m_1 \leq 4, m_3 \leq 6\}$, with same image $\{g_3\}$ are generators, i.e. there does not exist a subset of these itemsets with same image. However, their corresponding interval pattern are respectively $c = \langle[4,4],[7,9],[4,5]\rangle$ and $d = \langle[4,4],[7,9],[4,6]\rangle$ and we have $d \sqsubseteq c$, while $c^\square = d^\square$, hence c is not an interval pattern generator. This is due to the fact $m_3 \leq 6$ is a redundant condition with respect to $m_3 \leq 5$, the only IS-items that differ from N_2 to N_1.

Interval patterns with null support. The fact that the closed interval pattern of null support does not exists can be seen in the IS-context: consider the empty set of objects $\emptyset$, one has $\emptyset' = N$, hence no intervals, e.g. it contains items $\{m_1 \leq 4\} \in N$ and $\{m_1 \geq 5\} \in N$ that do not characterize an interval for m_1, since intervals are convex subsets of $\mathbb{R}$.

Moreover, IS-itemset generators cannot be found in the IS-scaled context. By definition, the IS-itemset generator of null support is unique and is given by $\emptyset$. Now, consider $c = \langle[6,6],[5,8]\rangle$: it does not exist a pattern $d \sqsubseteq c$ with same image, hence c is a generator. However, its corresponding itemset is $N_1 \subset N$, $N_1 = \{m_1 \geq 6, m_2 \geq 5\}$. Since $\emptyset \subset N_1$ and $\emptyset' = N_1' = \emptyset$, N_1 is not a generator in the binary table.

For these three reasons (local and global redundancy, and problem of null support), it should be not only more efficient to directly explore the search-space of interval patterns but also provide correctness. This is the aim of the next section.

5 Algorithms

In this section, we first detail a depth-first enumeration of interval patterns, starting with the most frequent one. Based on this enumeration, we design the algorithm *MinIntChange* for extracting frequent closed interval patterns (FCIP). This algorithm needs slight modifications to compute frequent interval pattern generators (FIPG), giving the algorithm *MinIntChangeG*.

5.1 Greedy enumeration

Consider firstly one numerical attribute of the example, say m_1. Its semi-lattice of intervals $(D_{m_1}, \sqcap)$ is composed of all possible intervals with borders in W_{m_1} and is ordered by the subsumption relation given in Section 3. The unique smallest element w.r.t. $\sqsubseteq$ is the interval with maximal size, i.e. $[4, 6] = [min(W_{m_1}), max(W_{m_1})]$ and maximal frequency (here 5). The basic idea of pattern generation lies in *minimal changes* for generating the direct subsumers of a given pattern. For example, two minimal changes can be applied to $[4, 6]$. The first consists in replacing the right border with the value of W_{m_1} immediately lower that 6, i.e. 5, for generating the interval $[4, 5]$. The second consists in repeating the same operation for the left border, generating the interval $[5, 6]$. Repeating these two operations allows to enumerate all elements of $(D_{m_1}, \sqcap)$. A right minimal change is defined formally as, given $a, b, v \in W_m$, $a \neq b$,

$$minChangeR([a, b]) = [a, v] \mid v < b, \nexists w \in W_m \text{ s.t. } v < w < b$$

while a left minimal change $minChangeL([a, b])$ is formally defined similarly. Minimal changes give direct next subsumers and implies a monotonicity property of frequency, i.e. support of $[a, v]$ is less or equal than support of $[a, b]$.

The generalization to several attributes is straightforward: for each pattern there are $2.|M|$ minimal changes for modifying the left and the right border for each attribute.

5.2 Non redundant enumeration

The greedy enumeration is based on minimal changes but does not prevent redundancy since a pattern can be generated several times. For example, considering the attribute m_1, interval $[5, 5]$ is generated two times: from $[4, 6]$ applying a right then a left minimal change, or applying a left then a right minimal change (indeed, we can see in Figure 2 that $[5, 5]$ subsumes two different patterns having a common subsumee $[4, 6]$).

To avoid redundancy, a lectic order on changes, or equivalently on patterns, is defined: after a right change, one can apply either a right or left change; after a left change one can apply only a left change. Figure 3 shows the depth-first traversal (numbered arrows) of diagram of $(D_{m_1}, \sqcap)$. Backtracks occur when an interval of the form $[w, w]$ is reached ($w \in W_{m_1}$), or no more change can be applied. Therefore, generated elements form a tree traversed depth first.

This pattern generation can be seen as a classical enumeration used by depth-first algorithms in data-mining. Indeed, each minimal change is the interpretation of an IS-item. Recall that IS-items are of the form "$m \leq w$" or "$m \geq w$". Applying a change $minChangeR([a, b]) = [a, v]$ to a interval pattern is equivalent to add the IS-item "$m \leq v$" in a corresponding IS-itemset. Dually, $minChangeL([a, b]) = [v, b]$ consists in the IS-item "$m \geq v$". These IS-items characterizing minimal changes are drawn on Figure 3. This figure accordingly represents a prefix-tree, factoring out the effort to process common prefixes or minimal changes.

Therefore, the lectic order can be also expressed in terms of IS-items. Any IS-item containing the symbol $\leq$ precedes any IS-item containing $\geq$. Secondly, if both IS-items contains $\leq$, the one with the largest value w precedes the other one. Dually, if both IS-items contains $\geq$, the one with the smallest value w precedes the other one. Notice that IS-items having the form "$m \leq max(W_m)$" or "$m_1 \geq min(W_m)$" are not considered since they do not characterize minimal changes.

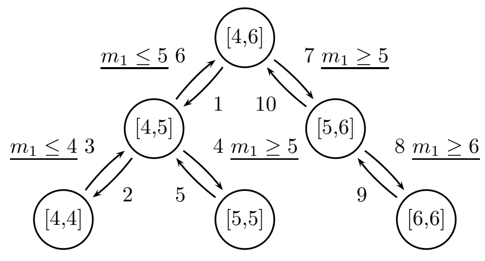

[4,6]
$\underline{m_1 \leq 5}$ 6
7 $\underline{m_1 \geq 5}$
1
10
[4,5]
[5,6]
$\underline{m_1 \leq 4}$ 3
4 $\underline{m_1 \geq 5}$
8 $\underline{m_1 \geq 6}$
2
5
9
[4,4]
[5,5]
[6,6]

Figure 3: Depth-first traversal of $(D_{m_1}, \sqcap)$.

The generalization to several attribute is again straightforward. A lectic order is classically defined on numerical attributes as a lexicographic order, e.g. $m_1 < m_2 < m_3$. Then changes are applied as explained above for all attributes respecting this order. For example, after applying a change to attribute m_2, one cannot apply a change to attribute m_1 since $m_1 < m_2$. On the example of Table 1, considering that $\langle[4,5],[8,9],[5,8]\rangle$ was previously generated from a left minimal change of a pattern for attribute m_2, only three patterns can be generated in the next step, namely, $\langle[4,5],[9,9],[5,8]\rangle$ (change on m_2 left), $\langle[4,5],[8,9],[5,6]\rangle$ (change on m_3 right) and $\langle[4,5],[8,9],[6,8]\rangle$ (change on m_3 left).

5.3 Extracting frequent closed interval patterns

The pattern enumeration starts with the minimal pattern w.r.t $\sqsubseteq$ and generates its direct subsumers with lower or equal support. The next problem now is that minimal changes do not necessarily generate patterns with strictly smaller support. Therefore, we should apply changes until a pattern with different support is generated to identify a closed interval pattern (FCIP) but this would not be efficient.

However, applying a minimal change does not mandatory implies that resulting pattern has strictly smaller support. Therefore, we should apply changes until the support changes to flag a FCIP. This would be not efficient as it required to generate the whole set of frequent interval patterns. We adopt the idea of the algorithm *CloseByOne* [18]: before applying a minimal change, the closure operator $(.)^{\square\square}$ is applied to the current pattern, allowing to skip all equivalent patterns. Indeed, the minimal pattern d w.r.t. $\sqsubseteq$ is closed as it is given by $d = G^{\square}$. Applying a minimal change returns a pattern c with strictly smaller support, since $d \sqsubseteq c$ and d is closed. If c is frequent, we can continue, apply the closure operator and next changes in lectic order, allowing to completely enumerate all FCIP.

Example. We start from the minimal pattern $c = \langle[4,6],[7,9],[4,8]\rangle$. The first minimal change in lectic order is a right change on attribute m_1. We obtain pattern $d = \langle[4,5],[7,9],[4,8]\rangle$, and obviously $c \sqsubseteq d$. However, $d^{\square\square} = \langle[4,5],[7,9],[5,8]\rangle$, hence d is not closed. Next change will be applied to $d^{\square\square}$.

Since a FCIP may have several different associated generators, it can be generated several times. Still following the idea of *CloseByOne*, a canonicity test can be defined according to lectic order minimal changes: if a pattern d has been generated by a change at attribute $m_j \in M$, it is canonically generated iff d and $d^{\square\square}$ do not differ for any attribute $m_h \in M$ such as $m_h < m_j$. This test avoids lookup in memory (e.g. using an hashtable of FCIP).

Example. Given the minimal pattern $\langle[4,6],[7,9],[4,8]\rangle$ and the pattern obtained by minimal change on left border for attribute m_3, i.e. $d = \langle[4,6],[7,9],[5,8]\rangle$. We have $d^{\square\square} = \langle[4,5],[7,9],[5,8]\rangle$. We observe that d and $d^{\square\square}$ present a difference for attribute m_1, but d has been generated from a change on m_3. Since $m_1 < m_3$, $d^{\square\square}$ is not canonical and has already been generated (see previous example): it is no more necessary to apply minimal changes to $d^{\square\square}$. Since this FCIP has already been generated, the algorithm backtracks, indicated by $d^{\square\square} <_D d$ in the algorithm given below.

MinIntChange. The algorithm is initialized as follow. G is the set of objects. $G^{\square}$ is the most frequent pattern and minimal w.r.t $\sqsubseteq$. Two integers are used to indicate the current minimal change (attribute and border). A minimal frequency min_{supp} is also given.

Alg. 1 MinIntChange()

```
1: $FCIP = \emptyset$; // the FCIP set
2: process($G^{\square}$,0,0,$G$,$G^{\square}$);
```

Given a generated closed pattern d, the main procedure firstly checks whether d is frequent and tests canonicity. If one of these test fails, the algorithm backtracks. Otherwise the current pattern d is stored as being a FCIP not previously generated. Then, the algorithm applies minimal changes to d following the lectic order (from attribute n and border p), computes closure and the procedure is called again. The procedure backtracks when no more minimal changes to current FCIP can be applied. The notation $\delta_{n,l}(d)$ returns the left border of the interval describing attribute n in d while $\delta_{n,r}(d)$ returns its right border. The peusdo code of the procedures $minChangeRight(d,n)$ and $minChangeLeft(d,n)$ is not given for sake of simplicity. It consists in applying the minimal change as previously defined (see *minChangeR([a,b])*) but for a given attribute, namely n. Accordingly, 18 FCIP are extracted from Table 1 with $min_{supp} = 1$. Note that the CIP of null support cannot be extracted if the user specifies $min_{supp} = 0$. The algorithm operates a bounded number of $2|M| \times |FCIP|$ minimal changes. Complexity of minimal change procedure is $log(W_m)$, i.e. getting the next value in a previously sorted set. For each change, closure is computed. First operator $(.)^{\square}$ returns the image of d and requires

Alg. 2 process(c, n, p, A, d), c was generated at previous step with a minimal change on attribute m and border p (p=0 means right, p=1 means left), $A = c^{\square}$ and $d = c^{\square\square}$

```
  if $(|A| \leq min_{sup}$ or $d <_D c)$ then
    return;
  end if
  $FCIP \leftarrow FCIP \cup d$
  for $i = n$ to $|M|$ step 1 do
    if $\delta_{i,l}(d) = \delta_{i,r}(d))$ then
      continue;
    end if
    if $(i = att$ and $p = 1) =$ false then
      $patR \leftarrow minChangeRight(d, i)$
      process(patR, i, 0, $patR^{\square}$, $patR^{\square\square}$);
    end if
    $patL \leftarrow minChangeLeft(d, i)$;
    process(patL, i, 1, $patL^{\square}$, $patL^{\square\square}$);
  end for
```

to scan objects in G and test if their description subsumes d. Actually, it its not needed to scan the whole set of objects, but only those in the image of the previously generated closed pattern. The second operator $(.)^{\square}$ applies to a set of objects, and returns the convex hull of their description in $\mathbb{R}^{|M|}$, requiring only computations of minima and maxima on each dimension separately.

5.4 Mining interval patterns generators

We now adapt *MinIntChange* to extract FIPG. Indeed, applying the closure operator to a generated pattern is still important: for any FCIP d, a minimal change implies that the support of the resulting pattern c is strictly smaller than the support of d. Therefore, c is a good generator candidate of the next FCIP. However, when applying the closure to this candidate, "equivalent changes can be added" and are not necessary to store for the next generator. This is made clearer with an example.

Example. Consider the pattern $\langle [4, 5], [7, 9], [4, 8]\rangle$ obtained with a right minimal change on the smallest pattern w.r.t $\sqsubseteq$, and characterized by the IS-items "$m_1 \leq 5$". Now consider its closure, i.e. $\langle [4, 5], [7, 9], [4, 8]\rangle^{\square\square} = \langle [4, 5], [7, 9], [5, 8]\rangle$. The closure adds one change, namely "$m_3 \geq 5$". Actually, it can be shown that the changes "$m_1 \leq 5$" and "$m_3 \geq 5$" are equivalent as they characterizing the same image.

Since a generator is characterized by a smallest set of minimal changes as possible (having largest intervals in its equivalence class), we should not consider the changes "added" by the closure. This can also be understood with Propositions 4.2 and 4.3.

At each step of the depth-first enumeration is generated a FGIP candidate. We know that it has no subsumers in its branch with same support. However, it could exist a branch with another FGIP with same image and resulting from less changes. Regarding to the lectic order on minimal changes, and already suggested in the binary case in [7], we should use a reverse traversal of the tree, see Figure 4. Therefore, if such pattern exists, i.e. the current candidate is not a generator, it has already been generated with few minimal changes. In this case, the algorithm backtracks: these two patterns have the same closure, hence the same minimal change will be used to build next candidate.

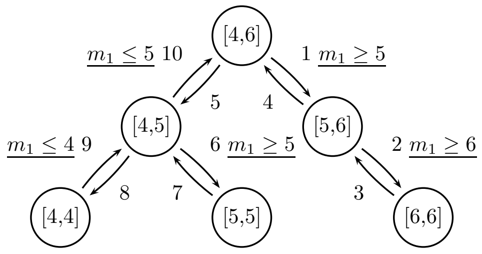

[4,6]
$\underline{m_1 \leq 5}$ 10
1 $\underline{m_1 \geq 5}$
5
4
[4,5]
[5,6]
$\underline{m_1 \leq 4}$ 9
6 $\underline{m_1 \geq 5}$
2 $\underline{m_1 \geq 6}$
8
7
3
[4,4]
[5,5]
[6,6]

Figure 4: Reverse pre-order traversal of $(D_{m_1}, \sqcap)$.

MinIntChangeG. At the initialization step, we start from the minimal pattern d. This pattern d is both closed and generator, i.e. $d = G^\square$ while any change would also change its support. d is stored as FCIP and FGIP. At a given step, if the generator candidate is actually a generator (see details after) and is frequent, the FCIP is used to characterize the next change. This change is applied to the FGIP to obtain the new candidate, the closure operator is applied to obtain its closure. Next step is called with resulting FCIP and the new FGIP. This means that a FGIP is characterized by a minimal set of changes (branches in the tree), while the FCIP is characterized by the maximal set of changes (branches plus changes added by successive closures). Notice that the canonicity test cannot be used anymore, since a FCIP may have several generators, characterized by different minimal sets of changes.

Alg. 3 MinIntChangeG

```
    $FIPG = \emptyset$;
    processGen(0,0,$G$,$G^\square$,$G^\square$);
```

Fast subsumption checking with hastable. To test whether a candidate is a generator, we use the same technique as in the algorithm *Charm* [27]. *MinIntChange* hashes the FIPG upon their image. In the testing of a candidate d, the entire list corresponding to its hash value $h(d)$ is retrieved. If there is a FGIP c in the list with same support and such that $c \sqsubseteq d$, d is discarded, otherwise d is declared a FIPG and hashed.

Fast subsumption checking with a trie. A second possibility uses the trie structure (see e.g. [6] for more details). Each word of the trie is the image of a FCIP, and a list of its generators its attached. When testing whether a candidate is a generator or not, we look in the trie for its corresponding image (word) and only test the generators associated to this word. If one of them is subsumed by the candidate, the candidate is discarded, otherwise added to the list. Whereas this solution may be more efficient, it requires more storage space. Most importantly, it allows to associate any FIPG to its closure, answering to the problem 3.

Alg. 4 processGen(n, p, A, d, cand): *cand* is the current candidate, $cand^{\square} = A$, $A^{\square} = d$

```
    if $|A| \leq min_{sup}$ or addCandidate(gen) = false then
2:      return;
    end if
4:  $FIPG = FIPG \cup cand$;
    for i=$|M|$ to n step - 1 do
6:      if $\delta_{o,l}(d) = \delta_{o,r}(d)$ then
            continue;
8:      end if
        $clone \leftarrow gen$
10:     $\delta_{i,l}(clone) \leftarrow \delta_{i,l}(minChangeLeft(d, i))$;
        process(i, 1, $clone^{\square}$, $clone^{\square\square}$,$clone$);
12:     if $(i = att$ and $p = 1) =$ false then
            $clone \leftarrow cand$
14:         $\delta_{i,r}(clone) \leftarrow \delta_{i,r}(minChangeRight(d, i))$;
            process(i, 0, $clone^{\square}$, $clone^{\square\square}$,$clone$);
16:     end if
    end for
```

Finally, notice that this algorithm is not able to extract generators with null support. Indeed, next changes are obtained from the previous FCIP. Since the FCIP with null support does not exist, we cannot find its generators. A basic solution, not investigated in this paper, is to use the non-redundant enumeration style when a generator of support one is produced.

6 Experiments

We evaluate here the performance of the algorithms designed in the previous section, namely *MinIntChange*, *MinIntChangeG-h* with auxiliary hashtable and *MinIntChangeG-t* with auxiliary trie. We also compare the performance with the itemset-mining algorithm *Charm* [27] for extracting closed IS-itemsets, and *TalkyG* [26] for extracting IS-itemsets generators. Indeed, *TalkyG* also considers a depth-first reverse traversal of the prefix tree of binary attributes, while also using a hashtable for fast subsumption checking. In the latter case, we also study the global redundancy effect. The three algorithms designed in this paper are implemented in Java. We used the *Coron System* [1] to experiment both *Charm* and *TalkyG*. Experiments are conducted on a 2.50Ghz machine with 16GB RAM running under Linux 2.6.18-92.e15.

Characteristics of the datasets used are given in Table 3 [11]. As the number of patterns depends on the number of unique attribute values in the dataset, we choose datasets with increasing size and number of attribute values in average, the worst case being when each attribute has a different value for each object (AP). The first experiments compare the algorithm *MinIntChange* for extracting FCIP and *Charm* for extracting equivalent frequent closed IS-itemsets in corresponding interordinally scaled context. Each execution time is the mean of five runs and is given in milliseconds in Table 4, for different datasets and minimum support. We do not compare memory usage here: *MinIntChange* uses

Dataset	Objects	Attributes	avg. $\lvert W_m \rvert$
Bolts (BL)	40	8	17
Basketball (BK)	96	5	62
Airport (AP)	135	5	135

Table 3: Datasets [11].

a canonicity test for pruning, while *Charm* uses an auxiliary hashtable. Note that the number of FCIP explodes when the support is low with dataset AP (worst case). The Table 4 shows that *MinIntChange* outperforms *Charm*. The table entry NA means that the computation was intractable.

minSupp	MinIntChange	Charm	$\lvert FCIP \rvert$
BL			
90%	< 50	< 50	112
80%	< 50	< 50	1,130
50%	**252**	658	32,107
25%	**1,215**	10,517	171,192
2%	**1,905**	42,679	272,223
AP			
90%	222	**190**	12,419
80%	**4,595**	24,309	346,741
50%	**145,939**	33,410,268	16,214,345
20%	**448,934**	NA	66,879,365

Table 4: Execution time for extracting FCIP (in ms).

We now extract frequent interval pattern generators (FIPG) with *MinIntChange-h* and *MinIntChange-t*. We also extract frequent itemsets generators (FISG) in corresponding binary data after interordinal scaling with the algorithm *TalkyG*. Table 5 gives the execution time (mean of five runs) for each algorithm, and the proportion of globally non-redundant FISG. Since *MinIntChange-t* provides each generator with its closure, the table accordingly gives the number of generators per closed interval pattern (last column). We first remark that the global redundancy effect discards the use of binary data, e.g. only 11% of the FISG are actually FIPG in dataset BL with a minimum support of 25%. This worsens the different execution times of *TalkyG*. Secondly, the algorithm *MinIntChangeG-t* outperforms *MinIntchangeG-h*. However, it uses more memory (not detailed here) since it stores each closed set of objects as a word in the trie, and to each word the list of FIPG associated to this closed set. Meanwhile, *MinIntChangeG-h* requires less memory since storing the FIPG only in an hashtable. Finally, we remark that the less frequent a FCIP is, the more its equivalence class has minimal elements (FIPG).

7 Discussion and conclusion

We presented a study on the characterization and the extraction of frequent closed interval patterns and their associated generators from numerical data.

	TalkyG	MinIntChangeG-h	MinIntChangeG-t	$\lvert FIPG \rvert$	$\lvert FISG \rvert$	$\frac{\lvert FIPG \rvert}{\lvert FISG \rvert}$	$\lvert FCIP \rvert$	$\frac{FIPG}{FCIP}$
BL								
90%	< 50	< 50	< 50	176	194	90%	112	1.57
80%	< 50	< 50	< 50	1952	2823	69%	1130	1.73
50%	9,031	1212	**529**	66,350	222,088	29%	32,107	2
25%	1,349,110	27,988	**3,893**	411,442	3,559,419	11%	171,192	2.4
2%	NA	438,214	**24,141**	1,165,824	NA	NA	272,223	4.3
BK								
90%	1,164	1,268	**1,207**	67,737	75,058	84%	48847	1.3
85%	65,967	26,154	**12,139**	554,956	799,574	69%	403,562	1.37
80%	1,470,000	512,126	**107,700**	2,730,812	4,787,490	57%	1,938,984	1.40

Table 5: Execution time for extracting FIPG and global redundancy evaluation.

For this task, we designed the algorithms *MinIntChange* and *MinIntChangeG*, our main contribution. These algorithms are reusable for other kind of data, for which a closure operator is defined (e.g. graphs in pattern structure [8]) and a minimal change operation is defined (e.g. adding an edge to a graph pattern). The main drawback of the algorithms lies in their poor scalability when the number of different attribute values is large compared to the number of objects. However, as stated in [15] for unfrequent closed interval pattern extraction, one can easily embed monotone constraints on the lattice structure of these patterns (e.g. minimal/maximal size of one or several intervals). Indeed, intervals with too large size tend to be frequent but not interesting, whereas small intervals may have too small support [25]. We dedicated this problem in [14], in the field of information fusion, by introducing a similarity relation between interval patterns. A second solution explored in [15] with effective results in gene expression data analysis, is to reduce the number of different attribute values before the mining task, e.g. rounding values. For example, the last attribute of the basket ball dataset (BK) describes the points per minutes of a player: a double value with four digits after the comma, e.g. 0.5885. One can round this value to two digits after the coma considering that this loss of information is not significant, making the mining possible with large datasets.

We also showed that mining equivalent binary data (encoding all possible intervals) is not efficient since these data suffers of redundancy. Indeed, classical itemset mining algorithms generally do not consider a semantic associated to binary attribute labels. That was also a contribution to show that pattern structures and associated closure operator provide a simple and elegant framework to consider numerical data.

Taking into account missing values is a perspective of research, while fault-tolerant interval patterns should be studied, possibly strongly reducing their number (see e.g. [5] for the binary case). Two applications in which interval pattern generators may be useful are proposed.

Generators are preferable to closed patterns. According to the version of minimum description length principle (MDL) of [10], the best hypothesis to explain a dataset is the one minimizing the sum of (i) the length in bit of the description of the hypothesis, and (ii) the length of the data description when encoded with the help of the hypothesis. The authors of [20] recalled how the MDL principle favors generators. Consider an equivalence class of itemset in binary data. The maximal element, i.e. closed itemset, has higher cardinality, while generators have smallest cardinality. Therefore, the generators with minimal cardinality are best hypothesis to describe the same set of objects. The same holds for interval patterns, modulo the notion of minimality: best patterns are those minimal w.r.t. the subsumption relation on patterns, i.e. patterns with largest interval describing a same set of objects. According to [20], interval pattern generators provide better hypothesis, and seem useful for numerical classification problems, i.e. explaining the resulting cluster description, since usually, the bounding box of object descriptions (a closed interval pattern), is considered.

Interval patterns for k-anonymity. To preserve privacy in a dataset, object identifiers can be removed, e.g. names. However, some combinations of attributes such as birth date and ZIP code possibly allow to identify a unique individual. An important method for de-identification is the method of k-

anonymity [2]. A basic idea is to reduce the granularity of data descriptions in such a way that a unique individual cannot be distinguished among at least $(k-1)$ individuals. For numerical attributes, a solution is to "generalize" the attribute values to a range, reducing the granularity, e.g. replacing the age 23 by an interval $[21, 24]$, see e.g. [23]. Now consider an individual $g \in G$ in a numerical dataset as described in the present paper. The description $\delta(g) \in (D, \sqsubseteq)$ is composed of degenerate intervals, and is closed. The information brought by one of its generators (with larger intervals) is as follows: this generalization is not sufficient enough to not uniquely identify the individual. One should therefore consider a smaller generator w.r.t. $\sqsubseteq$ depending on the cardinality of its image, and can replace the individual description this generator. This operation is a projection of the pattern searchspace.

Reproducible Results Statement:

For the interest of the community, algorithm source codes and datasets can be found at http://www.loria.fr/~kaytouem/SDM11.

References

[1] http://coron.loria.fr.

[2] C. C. Aggarwal and P. S. Yu. *Privacy-Preserving Data Mining: Models and Algorithms.* Springer, 2008.

[3] Y. Bastide, N. Pasquier, R. Taouil, G. Stumme, and L. Lakhal. Mining minimal non-redundant association rules using frequent closed itemsets. In *Computational Logic*, pages 972–986, 2000.

[4] R. Bendaoud, A. Napoli, and Y. Toussaint. Formal concept analysis: A unified framework for building and refining ontologies. In *Knowledge Engineering: Practice and Patterns (EKAW)*, LNCS 5268, pages 156–171, 2008.

[5] J. Besson, C. Robardet, and J.-F. Boulicaut. Mining a new fault-tolerant pattern type as an alternative to formal concept discovery. In *ICCS*, LNCS 4068, pages 144–157. Springer, 2006.

[6] F. Bodon and L. Rónyai. Trie: An alternative data structure for data mining algorithms. *Mathematical and Computer Modelling*, 38(7-9):739 – 751, 2003.

[7] T. Calders and B. Goethals. Depth-first non-derivable itemset mining. In *SIAM International Conference on Data Mining*, 2005.

[8] B. Ganter and S. O. Kuznetsov. Pattern structures and their projections. In H. S. Delugach and G. Stumme, editors, *ICCS*, volume 2120 of *Lecture Notes in Computer Science*, pages 129–142. Springer, 2001.

[9] B. Ganter and R. Wille. *Formal Concept Analysis.* Springer, 1999.

[10] P. D. Grünwald, I. J. Myung, and M. A. Pitt. *Advances in Minimum Description Length: Theory and Applications.* The MIT Press, 2005.

[11] H. A. Guvenir and I. Uysal. Bilkent university function approximation repository. http://funapp.cs.bilkent.edu.tr, 2000.

[12] J. Han, J. Pei, and Y. Yin. Mining frequent patterns without candidate generation. In W. Chen, J. F. Naughton, and P. A. Bernstein, editors, *SIGMOD Conference*, pages 1–12. ACM, 2000.

[13] D. Hand, H. Mannila, and P. Smyth. *Principles of Data Mining.* The MIT Press, Cambridge (MA), 2001.

[14] M. Kaytoue, Z. Assaghir, A. Napoli, and S. O. Kuznetsov. Embedding tolerance relations in formal concept analysis for classifying numerical data. In *19th Conference on Information and Knowledge Management (CIKM).* ACM, 2010.

[15] M. Kaytoue, S. O. Kuznetsov, A. Napoli, and S. Duplessis. Mining gene expression data with pattern structures in formal concept analysis. *Information Sciences*, In Press, Corrected Proof, 2010.

[16] S. O. Kuznetsov. Galois connections in data analysis: Contributions from the soviet era and modern russian research. In *Formal Concept Analysis*, volume 3626 of *LNCS*, pages 196–225. Springer, 2005.

[17] S. O. Kuznetsov. Pattern structures for analyzing complex data. In *Rough Sets, Fuzzy Sets, Data Mining and Granular Computing*, LNCS (5908), pages 33–44. Springer, 2009.

[18] S. O. Kuznetsov and S. A. Obiedkov. Comparing performance of algorithms for generating concept lattices. *J. Exp. Theor. Artif. Intell.*, 14(2-3):189–216, 2002.

[19] F. Le Ber, M. Benot, C. Schott, J.-F. Mari, and C. Mignolet. Studying crop sequences with CarrotAge, a hmm-based data mining software. *Ecological Modelling*, 191(1):170–185, 2006.

[20] J. Li, H. Li, L. Wong, J. Pei, and G. Dong. Minimum description length principle: Generators are preferable to closed patterns. In *Innovative Applications of Artificial Intelligence Conference.* AAAI Press, 2006.

[21] N. Pasquier, Y. Bastide, R. Taouil, and L. Lakhal. Pruning closed itemset lattices for association rules. *International Journal of Information Systems*, 24(1):25–46, 1999.

[22] M. Rouane-Hacene, M. Huchard, A. Napoli, and P. Valtchev. A proposal for combining formal concept analysis and description logics for mining relational data. In *International Conference on Formal Concept Analysis*, LNCS 4390, pages 51–65. Springer, 2007.

[23] P. Samarati. Protecting respondents identities in microdata release. *Knowledge and Data Engineering, IEEE Transactions on*, 13(6):1010 –1027, 2001.

[24] A. Soulet and B. Crémilleux. Adequate condensed representations of patterns. *Data Min. Knowl. Discov.*, 17(1):94–110, 2008.

[25] R. Srikant and R. Agrawal. Mining quantitative association rules in large relational tables. In H. V. Jagadish and I. S. Mumick, editors, *SIGMOD Conference*, pages 1–12. ACM Press, 1996.

[26] L. Szathmary, P. Valtchev, A. Napoli, and R. Godin. Efficient vertical mining of frequent closures and generators. In *IDA*, LNCS (5772), pages 393–404. Springer, 2009.

[27] M. J. Zaki and C.-J. Hsiao. Charm: An efficient algorithm for closed itemset mining. In R. L. Grossman, J. Han, V. Kumar, H. Mannila, and R. Motwani, editors, *SDM*. SIAM, 2002.

Contents

Éditeur
INRIA - Domaine de Voluceau - Rocquencourt, BP 105 - 78153 Le Chesnay Cedex (France)
http://www.inria.fr
ISSN 0249-6399