Edit Operations on Lattices for MDL-based Pattern Summarization

Keisuke Otaki[1,2] and Akihiro Yamamoto[1]

[1] Department of Intelligence Science and Technology,
Graduate School of Informatics, Kyoto University, Japan
ootaki@iip.ist.i.kyoto-u.ac.jp, akihiro@i.kyoto-u.ac.jp
[2] Research Fellow of the Japan Society for the Promotion of Science

Abstract. The closedness, a fundamental conception in FCA, is also important to address the pattern redundancy problem in pattern mining, where some enumerated patterns subsume others and they are redundant. Although many efficient mining algorithms have been proposed, finding characteristic and easy to interpret subsets of the whole enumerated patterns, called the *pattern summarization problem*, is still challenging. A well-used Information Theory-based criterion; the *Minimum Description Length* (MDL) principle helps us to tackle the problem, but it requires us to design the MDL evaluation from scratch according to types of patterns. In this paper we propose a new framework applicable to various patterns using lattices, which are beneficial to formulate the MDL evaluation. A key idea is revising an existing model by using *edit operations* defined on lattices among concepts on them, which enables us to consider additional information such as background knowledge and helps us to design the MDL evaluation. We experiment our method to see that our proposal helps us to obtain informative results from the whole sets, and confirm that our model is applicable to various patterns.

Keywords: formal concept analysis, lattice structures, edit operations, pattern summarization, minimum description length principle

1 Introduction

Knowledge Discovery using binary feature vectors is an important subject in data mining, where a binary value represents whether or not an object keeps some feature. A way to construct such vectors is using *pattern mining*: For a datum d and an enumerated pattern p, we compute a binary value $b_{d,p} = 1$ if and only if p occurs in d. Since the frequent itemset mining problem was first proposed by Agrawal and Srikant [2], various patterns and algorithms have been studied [1]. We can exploit obtained vectors as representations of data within several Machine Learning methods, and tasks such as clustering and classification can be tackled with the vectors. Therefore pattern mining is an essential tool.

Although many efficient mining algorithms have been studied, the *pattern redundancy problem* is still inevitable, where some patterns subsume others, enumerated patterns are not independent, and obtained vectors contain redundant

Local Proceedings also appeared in ISBN 978-84-606-7410-8, Universidad de Málaga (Dept. Matemática Aplicada), Spain.

values. Considering the closedness, a main conception in FCA, is a fundamental way to resolve the problem and an example of the closedness used in pattern mining is closed itemsets [12], known as compact representations of itemsets.

The redundancy problem, however, still remains after considering the closedness only. To select a subset of the whole set of enumerated patterns, the *Minimum Description Length* (MDL) principle has attracted much attention [7, 8, 14–16, 18]. An important idea is to *represent databases with codes in a loss-less manner*, and to evaluate its difficulty by the length of codes used. For example, the Krimp algorithm [18] is a fundamental method for itemsets. Other methods for sequences and large graphs can be seen in [16] and [8], respectively. According to patterns, we need to design an evaluation method from scratch and it makes theoretical analyses difficult. It is also hard to integrate MDL-based methods with knowledge resources although various *side information* (e.g., corpora, ontology, etc.) are helpful to select enumerated patterns.

Developing methods applicable to various patterns in a unified framework and possible to consider side information is, therefore, an important but remaining problem. We in this paper present preliminary results of our new lattice-based framework based on an existing MDL-based method to enhance its utility for further developing of such methods to overcome problems above. Our key ideas are introducing edit operations on lattices (in Section 3.1) for the loss-less representation (by Proposition 1, Lemmata 1 and 2), and evaluating its difficulty by extending an existing formulation (in Section 3.2). To investigate the application possibility of our framework, we apply it to pattern concept lattices (in Section 4). We experiment them using common benchmark datasets (in Section 5). Our contributions are summarized below.

- *A new lattice-based framework*; it is extended from an existing model (code-tables in Section 2.2) by using terminologies from lattices.
- The *equivalency* for *closed itemsets* (by Proposition 1).
- *Pattern structured-based generalization*; *GenSet*, a generalization of itemsets using pattern structures (in Section 4) as an example of other lattices.
- *Empirical evaluation* using some benchmark datasets (in Section 5).

2 Preliminary

We outline Formal Concept Analysis (FCA) and Frequent Closed Itemset Mining (FCIM) for our new lattice-based framework, summarized in Table 1. We explain an idea of the MDL-based algorithm for itemsets, named the Krimp algorithm.

2.1 Basic Concepts

Formal Concept Analysis [5] Let $K = (G, M, I)$ be a *context*, where G and M are sets of objects and attributes, and $I \subseteq G \times M$ is a binary relation. A *Galois connection* between G and M, denoted shortly by $(\cdot)'$, is defined as $A' = \{m \in M \mid (g, m) \in I \text{ for all } g \in A\}$ for $A \subseteq G$ and $B' = \{g \in G \mid (g, m) \in$

Table 1: The correspondence between FCA and FCIM.

	FCA [5]	FCIM with a parameter θ [12]
objects	G; a set of objects	$\mathcal{D}$; a set of transactions
attributes	M; attributes	$\mathcal{I}$; items
functions	$\{(\cdot)', (\cdot)'\}$	$\{f, g\}$
task	find all concepts	find all itemsets $I \subseteq \mathcal{I}$ satisfying $freq(I) \geq \theta$

I for all $m \in B\}$ for $B \subseteq M$, respectively. A pair (A, B) is a *concept* if $A' = B$ and $B' = A$, where A and B are called the *extent* and the *intent*, and we denote them by $A = \mathrm{Ext}(c)$ and $B = \mathrm{Int}(c)$, respectively. For an object $g \in G$, the concept $(\{g\}'', \{g\}')$ is the *object concept* of g, denoted by γg.

Computing all concepts is an important task in FCA, and we denote the set of all concepts by $\mathfrak{B}(K)$. For two concepts (A, B) and (C, D), the partial order $\preceq$ is introduced by $A \subseteq C$, and $\langle \mathfrak{B}(K), \preceq \rangle$ becomes a complete lattice. We represent it by $\mathfrak{B}$ if it is understood from the context. For a concept $c \in \mathfrak{B}$, we define the *upset* of c, denoted $\uparrow c$, by $\{d \in \mathfrak{B} \mid c \preceq d\}$[3]. In addition we define the *direct upset* $\Uparrow c$ by $\{d \in \uparrow c \mid$ there exists no $e \in \uparrow c$ s.t. $e \preceq d\}$.

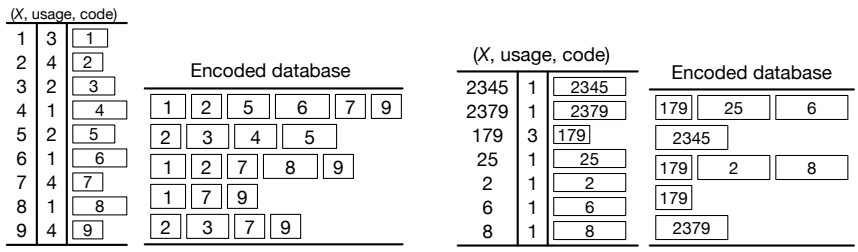
Frequent Closed Itemset Mining [12] Let $\mathcal{I}$ be the set of items. A set $I \subseteq \mathcal{I}$ of items is called an *itemset*, and it is also called a *transaction*. A *database* $\mathcal{D} = \{t_1, \ldots, t_N\}$ is a set of transactions. The *frequency* $freq(I)$ of an itemset I is defined as $freq(I) = |\{t \in \mathcal{D} \mid I \subseteq t\}|$. For a parameter θ, the frequent itemset mining (FIM) problem is the problem to find all itemsets satisfying $freq(I) \geq \theta$. We denote the set of all frequent itemsets by $\mathcal{F}$.

For *closed itemsets*, two functions f and g are defined as $f(T) = \{i \in \mathcal{I} \mid \forall t \in T, i \in t\}$ for $T \subseteq \mathcal{D}$ and $g(I) = \{t \in \mathcal{D} \mid \forall i \in I, i \in t\}$ for $I \subseteq \mathcal{I}$. An itemset I is *closed* if and only if $f(g(I)) = I$. The *frequent closed itemset mining* (FCIM) problem is to find all frequent and closed itemsets from $\mathcal{D}$ with respect to θ. It is easy to see that two functions f and g work in the same manner of $(\cdot)'$ in FCA[4]. We denote the set of all frequent *closed* itemsets by $\mathcal{CF}$.

The closed itemsets aim at reducing the number of all frequent itemsets enumerated in the *naïve* FIM problem, and many efficient algorithms have been studied [13, 17]. However the number of enumerated itemsets is still large in applications. They are often redundant because some patterns subsume others, which causes the *pattern redundancy problem*: *The enumerated patterns are too large and redundant to exploit them in applications although the closedness is considered*. In application, we often need only a small subset that characterizes all the enumerated itemsets, which are easy to interpret. We call such extraction problem *pattern summarization* (also known as *pattern set mining* [6, 15, 18]).

[3] This set $\uparrow c$ is also called the *principal filter* in partially ordered sets.

[4] We prepare *ids* of transactions and the set $\mathcal{D}' = \{(i_1, t_1), \ldots, (i_N, t_N)\}$. By constructing $I = \{(i_k, j) \mid 1 \leq k \leq N, j \in t_k\}$ from $\mathcal{D}'$, the FCIM problem of $\theta = 1$ is equivalent to computing all intents of concepts from $K = (\{i_1, \ldots, i_N\}, \mathcal{I}, I)$.



(a) The standard code-table ST (left) and the cover of $\mathcal{D}$ by ST (right).

(b) A code-table CT (left) and the cover of $\mathcal{D}$ by CT (right).

Fig. 1: The cover of $\mathcal{D} = \{125679, 2345, 12789, 179, 2379\}$ (from Example 1).

2.2 Two-part MDL Principle and The Krimp Algorithm

Out of various criteria for the selection, the *two-part MDL principle* [7] is often adopted for itemsets, where *code-tables* are well-studied models. Databases (or contexts) are encoded and represented by (binary) codes using code-tables.

Definition 1 (Code-tables [18]). A *code-table* CT consists of two columns for *itemsets* and *codes*. Formally $CT = \{(X_1, c_1), (X_2, c_2), \dots\}$ is a set of pairs, where $X_i \subseteq \mathcal{I}$, $c_i \in \mathbf{C}$, and $\mathbf{C}$ be the set of (binary) codes.

For the convenience of our discussion, in the following, we regard code-tables as *sets of itemsets*: For example $X \in CT$ means $(X, c_X) \in CT$ and c_X is denoted by $code_{CT}(X)$. We define $CT^- = \{(X, c_X) \in CT \mid |X| > 1\}$. The *standard code-table* ST consists of all *singleton itemsets* over $\mathcal{I}$. Let us first introduce examples, where sets are written as sequences (e.g. $\{1, 7, 9\}$ is represented as 179).

Example 1. Let $\mathcal{I} = \{1, \dots, 9\}$ and $\mathcal{D} = \{125679, 2345, 12789, 179, 2379\}$. In Fig. 1a and 1b, the left table is a code-table and the right one is a database represented by codes, where *rectangles* represent codes (e.g., the rectangle 7 means the code of the itemset 7). In Fig. 1a, the transaction 12789 is encoded separately. In Fig. 1b ($CT = ST \cup \{25, 179, 2345, 2379\}$), it is done by $179 \cup 2 \cup 8$.

The process of representing databases using code-tables is called the *cover*. Both frequent itemsets (e.g., 179) and non-frequent ones (e.g., 2345 or 2379) are used and the MDL principle takes a balance among them. The property of the cover of transactions is described by *cover functions*.

Definition 2 (Cover function[5]). Let $\mathcal{CT}$ be the set of all possible code-tables over $\mathcal{I}$, $\mathcal{D}$ be a database, $t \in \mathcal{D}$ be a transaction, and $CT \in \mathcal{CT}$ be a code-table. We call the set of itemsets $\{X \mid (X, c_X) \in CT\}$ the *coding set* of CT, denoted by CS. A *cover function* on CT, $cover : \mathcal{CT} \times 2^{\mathcal{I}} \to 2^{CS}$, should satisfy:

C1. if $X, Y \in cover(CT, t)$ then it holds either $X = Y$ or $X \cap Y = \emptyset$, and

[5] This definition is modified from Definition 2 of [18] by using a coding set CS.

C2. the union of all $X \in \mathit{cover}(CT, t)$ equals to t, i.e., $t = \bigcup_{X \in \mathit{cover}(CT,t)} X$.

Code-tables in Fig. 1 contain additional information *usage*. Using coding methods in the Kolmogorov Complexity [9], codes are decided according to *how often itemsets are needed*: For $X \in CT$, $\mathit{usage}(X)$ is defined as $|\{t \in \mathcal{D} \mid X \in \mathit{cover}(CT, t)\}|$, which determines the least length of prefix codes required.

Example 2 (from Example 1). In Fig. 1a, $\mathit{cover}(ST, 179) = \{1, 7, 9\}$, and in Fig. 1b $\mathit{cover}(CT, 12789) = \{179, 2, 8\}$. It holds that $\mathit{usage}(179) = 3$.

If a transaction $t \in \mathcal{D}$ is encoded by a cover function $\mathit{cover}(\cdot, \cdot)$ and some CT, we say that t is *covered*. Then the MDL principle can be evaluated.

Definition 3 (Lemma 1, Definitions 4 and 5 in [18]). The two-part MDL principle says that a code-table $CT \in \mathcal{CT}$ is the *best* for $\mathcal{D}$ if it minimizes $L(\mathcal{D}, CT) = L(CT) + L(\mathcal{D} \mid CT)$, where $L(CT)$ is the length of the code-table CT, and $L(\mathcal{D} \mid CT)$ is that of $\mathcal{D}$ when CT is given. Following Definitions 1 and 2, terms $L(CT)$ and $L(\mathcal{D} \mid CT)$ for $L(\mathcal{D}, CT)$ are evaluated as follows:

$$L(CT) = \sum_{X \in CT \text{ if } \mathit{usage}(X) \neq 0} L(\mathit{code}_{ST}(X)) + |\mathit{code}_{CT}(X)|, \tag{1}$$

$$L(\mathcal{D} \mid CT) = \sum_{t \in \mathcal{D}} \sum_{X \in \mathit{cover}(CT,t)} |\mathit{code}_{CT}(X)|, \tag{2}$$

where $L(\mathit{code}_{ST}(X)) = \sum_{i \in X} |\mathit{code}_{ST}(\{i\})|$.

The Krimp algorithm [18] is an iterative algorithm updating a code-table $CT \in \mathcal{CT}$ greedy by adding a new frequent itemset $X \in \mathcal{F}$ with evaluating $L(\mathcal{D}, CT)$. Please refer to Algorithm 3 in [18], which we follow in experiments.

2.3 Problems

Models based on code-tables have been widely used in MDL-based studies. The cover function adopts the set union to represent transactions, and the subset operation to divide them as seen in Example 1. The process assumes that items are in the same granularity, but it is not the case in applications. It also disables us to allow for side information (i.e., our background knowledge and resources) about databases as well. Then our targeting problems are listed below.

Problem 1. The cover function has a drawback of the difficulty of its generalization because it implicitly uses set operations.

Problem 2. Code-tables cannot reflect side information of items (e.g., *hierarchy*, or *degree of importance* of items) because of implicitly used set operations.

That is we need to revise set operations used in the existing method. Our key idea is to adopt terminologies of lattices to overcome these problems. In Section 3, we revise the cover function with lattices to build our new framework. We apply it to *pattern concept lattices* by *pattern structures* in Section 4 to confirm that our lattice-based framework is applicable to various patterns.

3 Models by Edit Operations for Closed Itemsets

We provide our framework based on lattices. For discussions, we define the object concept corresponding to $t \in \mathcal{D}$ by $\gamma t = (g(t), t)$ from the relation in Table 1. Here we only consider closed itemsets.

3.1 Cover Using Edit Operations on Lattices

In the cover of $t \in \mathcal{D}$, t is recursively replaced with $t \setminus B_1, t \setminus (B_1 \cup B_2), \ldots$ by choosing mutually disjoint closed itemsets $B_1, B_2, \ldots$ until t gets $\emptyset$ in some order[6]. We now have the following.

Observation 1 The cover of $t \in \mathcal{D}$ can be represented as a *sequence* of the *deletion* of items: $t \mapsto t \setminus B_1, t \setminus B_1 \mapsto t \setminus (B_1 \cup B_2), \ldots, t \setminus (B_1 \cup \cdots \cup B_m) \mapsto \emptyset$.

Thus a transaction $t \in \mathcal{D}$ can be covered by also using the deletion of items. We use such deletions to construct a new cover based on lattices. On lattices, selections of B_i corresponds to selecting concepts from $\mathfrak{B}$, and intents of them can be stored in code-tables. With respect to the search space of the selection, the following holds by properties of lattices.

Observation 2 For a transaction $t \in \mathcal{D}$, only a sub-lattice corresponding to the upset $\uparrow \gamma t$ of the corresponding object concept γt of t should be considered.

Choosing concepts, however, is *not complete* for the *loss-less property* (i.e., C2 of Definition 2) because in general $\mathfrak{B}$ does not contain all singleton itemsets in intents of concepts. In contrast, by the discussion of closed itemsets [12], all frequent itemsets could be recovered from closed itemsets and lattices should contain all information required.

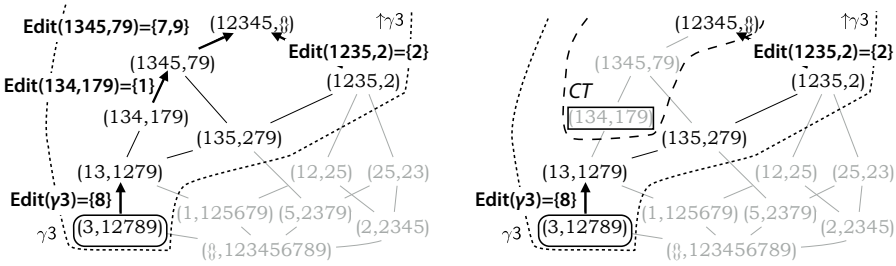
Our idea is extracting such hidden information on lattices as edit operations on lattice (e.g., the deletion of items in Observation 1), and represent $t \in \mathcal{D}$ by both the selections and edit operations. We first introduce *edit information*.

Definition 4 (Edit Information). For $c = (A, B) \in \mathfrak{B}$, the *edit information*, denoted by $\mathrm{Edit}(A, B)$, is defined as $\mathrm{Edit}(A, B) \equiv B \setminus \bigcup_{(C,D) \in \Uparrow(A,B)} D$. For a set $\mathcal{C}$ of concepts, it is extended by $\mathrm{Edit}(\mathcal{C}) = \bigcup_{(A,B) \in \mathcal{C}} \mathrm{Edit}(A, B)$.

For $c \in \mathfrak{B}$, $\mathrm{Edit}(c)$ indicates items which *cannot be represented by intents of concepts* except c itself even when all concepts in $\uparrow c \setminus \{c\}$ are chosen.

Example 3 (from Example 2). For $t_3 = 12789$, it holds that $\gamma 3 = (3, 12789)$ and $\mathrm{Edit}(\gamma 3) = \{8\}$. The item 8 *cannot be represented* by any concepts except $\gamma 3$. Therefore if $\gamma 3$ is not selected the item 8 *must be represented by another way.*

[6] The *standard cover order* is a basic one from the KRIMP algorithm: For an itemset $B \subseteq \mathcal{I}$, the *standard cover order* is the order of *the cardinality* $|B|$ *descending, the support freq*(B) *descending*, and *lexicographically ascending* [18]



(a) For $t_3 = 12789$ with ST (b) For $t_3 = 12789$ with CT

Fig. 2: Examples of the cover with edit operations for $\gamma 3 = (3, 12789)$ (i.e., t_3).

As another way, *edit operations* among intents of concepts are introduced to represent such information. Because $\mathfrak{B}$ contains the top $\top = (G, \emptyset)$[7] and $\mathrm{Int}(\gamma t) = t$, all information for covering t should be found in $\uparrow \gamma t$. We use edit information to capture such information and to explain edit operations on lattices (i.e., on intents of concepts). We encode edit information by tuples called *edit scripts*, and store them in *edit-tables* by analogy with code-tables.

Definition 5 (Edit scripts and Edit-tables). An *edit-table* ET consists of three columns: *kinds of operations*, arguments of operations (itemsets), and codes. That is, $ET = \{(t_1, X_1, c_1), \ldots, (t_m, X_m, c_m)\}$, where t_i indicates the type of operations, and both X_i and c_i is the same to code-tables. The tuple (t, X, c_X) is called an *edit script*. Let $\mathcal{E}$ be the set of all edit scripts.

Example 4 (from Example 3). Because $\mathrm{Edit}(\gamma 3) = \{8\}$, the item 8 should be represented by the deletion of the item 8. Let DEL represent the *deletion*[8]. A script $(\mathrm{DEL}, 8, c_8)$ with a code c_8 means the deletion of the item 8.

Definition 5 is general and abstract. For closed itemsets, however, we only need a specific kind of them. We define a function $\mathrm{Encode} : 2^{\mathcal{I}} \to 2^{\mathcal{E}}$ for $X \subseteq \mathcal{I}$ by $\mathrm{Encode}(X) = \{(\mathrm{DEL}, i, c_i) \mid i \in X, c_i \in \mathbf{C}\}$, where $\mathbf{C}$ is the set of codes.

Lemma 1. It holds that $\mathrm{Edit}(\uparrow \gamma t) = t$ for any $t \in \mathcal{D}$.

Proof If $\uparrow \gamma t = \{\gamma t, \top\}$, it holds that $\mathrm{Edit}(\uparrow \gamma t) = \mathrm{Edit}(\gamma t) = t$. Then $\mathrm{Encode}(\cdot)$ generates edit scripts of all items in t as the deletion of an item. By the induction on lattices (by $\Uparrow c$), we only need to confirm that for $c \in \mathfrak{B}$ all items which cannot be covered by intents of concepts in $\Uparrow c \setminus \{c\}$ can be represented by edit scripts. This is done by the definition of $\mathrm{Edit}(c)$ and properties of $\langle \mathfrak{B}(K), \preceq \rangle$. □

Lemma 1 guarantees that any $t \subseteq \mathcal{I}$ can be represented by using edit scripts when no concepts are chosen. Now the cover using lattices can be described using a function $cover_{\mathrm{L}}(\cdot, \cdot)$ by analogy with $cover(\cdot, \cdot)$. The following describes how to implement it instead of giving a new formal description of the function.

[7] If $B \neq \emptyset$ for the top $\top = (G, B)$ of $\langle \mathfrak{B}(K), \preceq \rangle$, we can insert a dummy $(G, \emptyset)$.
[8] The type is required to introduce several types of edit operations if needed.

Definition 6 (Revised Cover). The function $cover_{\mathrm{L}} : \mathcal{CT} \times 2^{\mathcal{I}} \to 2^{2^{\mathcal{I}}} \times 2^{\mathcal{E}}$ receives a code-table CT containing intents of selected concepts and a transaction t, and it returns a set of itemsets used (e.g., 179) and that of edit scripts.

For a code-table CT (See Fig. 2b), we first choose intents of concepts from CT in some order, and denote by $\mathcal{C}$ the set of selected concepts, which should be excluded from the search space $\uparrow \gamma t$. The remainder should be represented by edit scripts. Then $cover_{\mathrm{L}}(CT, t) = (\{\mathrm{Int}(c) \mid c \in \mathcal{C}\}, \mathrm{Encode}(\mathrm{Edit}(\uparrow \gamma t \setminus \uparrow \mathcal{C})))$.

Example 5 (from Example 3). It holds that $cover_{\mathrm{L}}(CT, t_3) = (\{179\}, \{(\mathrm{DEL}, 2, c_2), (\mathrm{DEL}, 8, c_8)\})$, representing the cover of t_3 in Fig. 2b. It holds that $cover_{\mathrm{L}}(ST, t_3) = \{\emptyset, \{(\mathrm{DEL}, 1, c_1), (\mathrm{DEL}, 2, c_2), (\mathrm{DEL}, 7, c_7), (\mathrm{DEL}, 8, c_8), (\mathrm{DEL}, 9, c_9)\}\}$, which simulates the cover of t_3 by singleton itemsets (in Fig. 2a).

If the set $\mathcal{C}$ in Definition 6 is the empty set $\emptyset$, the function $cover_{\mathrm{L}}(\cdot, \cdot)$ achieves the same result as the function $cover(\cdot, \cdot)$ with ST, supported by Lemma 1 (See Example 5). If some concepts are selected, the following lemma is important.

Lemma 2. Let $\mathcal{C}$ be the set in Definition 6 and $(\mathcal{X}, \mathcal{Y}) = cover_{\mathrm{L}}(CT, t)$. The set $\mathcal{Y}$ is complete to rerpesent $t \setminus \bigcup_{X \in \mathcal{X}} X$.

Proof From Definition 6, if a concept $c = (A, B)$ in $\uparrow \gamma t$ is selected and stored in CT (i.e., $c \in \mathcal{C}$), any item $i \in B$ is *not* represented by edit scripts in $\mathcal{Y}$ because there exists no concept $d \in \uparrow \gamma t \setminus \uparrow \mathcal{C}$ satisfying $i \in \mathrm{Edit}(d)$. It holds that $\{i \mid i \in \mathrm{Edit}(\uparrow \gamma t \setminus \uparrow \mathcal{C})\} = t \setminus \bigcup_{X \in \mathcal{X}} X$, and the function $\mathrm{Encode}(\cdot)$ ensures the lemma, i.e., the cover by $cover_{\mathrm{L}}(\cdot, \cdot)$ is valid when some concepts are chosen.

Together with the completeness in Lemma 1 and Lemma 2, our edit scripts are enough to represent any $t \in \mathcal{D}$ with selecting concepts in $\mathcal{C}$.

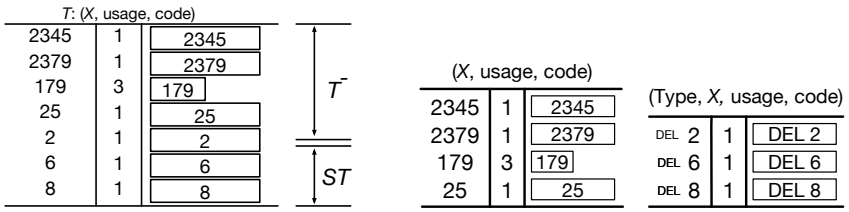
3.2 Model Evaluation Using Edit Operations

For our model (CT, ET), with reference to Section 2.2, the MDL principle $L(\mathcal{D}, CT, ET) = L(CT, ET) + L(\mathcal{D} \mid CT, ET)$ can be evaluated. For $L(CT, ET)$,

$$L(CT, ET) = \sum_{\substack{(X, c_X) \in CT \\ usage(X) \neq 0}} L(code_{ST}(X)) + |c_X| + \sum_{\substack{(t, X, c_X) \in ET \\ usage(X) \neq 0}} L(code_{\mathrm{OP}}(t)) + L(code_{ST}(X)) + |c_X|, \quad (3)$$

where $code_{\mathrm{OP}}(t)$ represents codes used for distinguishing *types of edit scripts*. For itemsets, $L(code_{\mathrm{OP}}(t)) = 0$. The first $\sum$ term of Equation 3 is the same to the KRIMP algorithm, and the second $\sum$ term is for encoding the edit-table ET.

The difficulty of representing databases, $L(\mathcal{D} \mid CT, ET)$, is evaluated by naturally generalizing the corresponding term of the KRIMP algorithm below.



(a) Code-table T ($= T^- \cup ST$) (b) Our model, two tables (CT, ET)

Fig. 3: A code-table T by the KRIMP algorithm and our two tables (CT, ET). Our table CT is equivalent to T^- and ET simulates ST.

$$L(\mathcal{D} \mid CT, ET) = \sum_{t \in \mathcal{D}} \left(\sum_{(X, c_X) \in \mathcal{X}} |c_X| + \sum_{(t, X, c_X) \in \mathcal{Y}} |c_X| \right) \text{for } (\mathcal{X}, \mathcal{Y}) = \mathit{cover}_{\mathrm{L}}(CT, t). \quad (4)$$

We illustrate an example in Fig. 3 to show how edit-tables intuitively work.

3.3 The Equivalency of Models for Closed Itemsets

For closed itemsets, we now have the following proposition.

Proposition 1 Let T be an obtained code-table by the KRIMP algorithm for the database $\mathcal{D}$ with some order on $\mathcal{CF}$. With the same order, our model can achieve a pair (CT, ET) satisfying $L(\mathcal{D}, T) = L(\mathcal{D}, CT, ET)$ for closed itemsets.

Proof (*Sketch)* At the beginning, it holds that $T = ST$ (i.e., $T^- = \emptyset$) and $CT = \emptyset$, where all transactions should be covered by singleton itemsets. From the definition of Encode$(\cdot)$ and Lemma 1, the proposition immediately holds. If a closed itemset $B \in \mathcal{CF}$ is selected and stored as $B \in T$, there exists a concept $(A, B) \in \mathfrak{B}$ and its intent B is stored in our table CT. Because we use the same order in the cover of $t \in \mathcal{D}$, for $(\mathcal{X}, \mathcal{Y}) = \mathit{cover}_{\mathrm{L}}(CT, t)$, it holds that $\mathcal{X} = \mathit{cover}(T, t)$, meaning that the *usage* in T^- and CT must be identical. The *usage* of edit scripts must be the same as well by the fact $\mathcal{X} = \mathit{cover}(T, t)$, Lemmata 1, 2, and the definition of Encode$(\cdot)$. □

Summary Our idea is to revise the cover in the KRIMP algorithm with lattices. Our framework divides code-tables into a part of intents of concepts (i.e., closed itemsets) and that of edit scripts simulating singleton itemsets. For *closed itemsets*, our model can achieve the same result as that obtained by the KRIMP algorithm (Proposition 1), which are supported by properties of lattices.

The advantage of our lattice-based model is that lattices help us to generalize the MDL-based evaluation for various patterns. For example, *pattern concept*

lattices [4] enable us to address the problem for several patterns with the MDL principle. The key concept for the generalization is to introduce edit operations on lattices according to patterns, encode them as edit scripts in edit-tables, where lattices become the search space in our framework. Lattices help us to consider the completeness of edit operations as well.

4 A Generalized Model via PSA

We apply our framework to *pattern concept lattices*. As an example of other lattices, we consider *GenSets*, a generalized version of itemset using *pattern structures*, which are introduced to analyze non-binary data [4]. We call methods using them for Knowledge Discovery *Pattern Structure Analysis* (PSA) below.

4.1 Pattern Structures and Generalized Sets

Pattern structures extend FCA using *meet semi-lattices*; a *meet semi-lattice* $(D, \sqcap)$ is a pair of a set D of *descriptions* representing fragments of objects as their features, and a meet operator $\sqcap$ between descriptions satisfying the associativity, the commutativity, and the idempotency. A partial order $\sqsubseteq$ of descriptions is induced by $\sqcap$ for $x, y \in D$ as $x \sqsubseteq y$ whenever $x \sqcap y = x$. A *pattern structure* $\mathbb{P}$ is a triple $\mathbb{P} = (G, (D, \sqcap), \delta)$, where G is a set of objects, $(D, \sqcap)$ is a meet semi-lattice, and a mapping $\delta : G \to D$ gives a descriptions for each object.

In PSA we use the following Galois connection $(\cdot)^\diamond$: $A^\diamond = \sqcap_{g \in A} \delta(g)$ for $A \subseteq G$ and $d^\diamond = \{g \in G \mid d \sqsubseteq \delta(g)\}$ for $d \in D$. A pair $(A, d) \in 2^G \times D$ is a *pattern concept* if and only if $A^\diamond = d$ and $d^\diamond = A$. A partial order among pattern concepts is defined as well, and *pattern concept lattices* can be constructed in a similar manner of FCA. Recent applications of PSA and lattices constructed by PSA can be seen in [3, 11]. We consider a slight modification of FCA by giving additional information to itemsets from the following scenario.

Example 6 (from Example 1). Let us consider $t_4 = 179$ of id i_4 and $t_5 = 2379$ of id i_5. We have $(i_4 i_5)' = 79$ computed by $i_4' \cap i_5' = 179 \cap 2379 = 79$. The remainders for each transaction are $1 = 179 \setminus 79$ and $23 = 2379 \setminus 79$, respectively. In this case, the existence of these items (i.e., 1 in t_4 and $2, 3$ in t_5) is *ignored* although they are useful for the representation of itemsets.

From this example, by using a meet semi-lattice, we introduce auxiliary numbers representing the *existence of items*, i.e., the number of items may be included at most in common, to enhance the expressiveness of itemsets.

Definition 7 (GenSets). We define $D = 2^{\mathcal{I}} \times \mathbb{N}$. For an itemset $B \subseteq \mathcal{I}$, we encode it by a pair $(B, 0) \in D$. The meet operator $\sqcap$ is defined as

$$(B_1, n_1) \sqcap (B_2, n_2) = (B_1 \cap B_2, \min(n_1 + |B_1 \setminus B_1 \cap B_2|, n_2 + |B_2 \setminus B_1 \cap B_2|)).$$

We call itemsets with additional numbers *GenSets* (*generalized sets* of items) [9].

Example 7. For two itemsets $t_4 = 179, t_5 = 2379$, we encode them as $d_1 = (179, 0)$ and $d_2 = (2379, 0)$, respectively. Then $d = d_1 \sqcap d_2$ is computed as $d = (B, n) = (79, 1)$, where $n = 1$ means that both itemsets t_4 and t_5 possibly have further one item. In other words, a GenSet $(79, 1)$ can be regarded as $\{7, 9, \star\}$ with a wildcard symbol $\star$ representing any item in $\mathcal{I}$.

We can interpret GenSets as a richer representation, and conjecture that they are helpful to capture characteristic features of itemsets. In other words, we now consider to take care of both the extent and the intent of concepts in evaluating the MDL principle. However, the KRIMP algorithm cannot deal with GenSets directly because the symbol $\star$ is not in $\mathcal{I}$ and it has completely different meaning. Therefore, applying the ordinal cover is not suited and a new formulation is required from scratch. In contrast, our lattice-based model can be applicable. We can evaluate the two-part MDL principle for GenSets by only considering the cover function $cover_{\mathrm{L}}(\cdot, \cdot)$ using edit operations including $\star$.

4.2 The two-part MDL principle for GenSets

We can evaluate the two-part MDL principle only by considering the function $cover_{\mathrm{L}}(\cdot, \cdot)$ using edit operations based on pattern concept lattices.

Models for GenSets For a GenSet $(B, n) \in D$, we need to encode n to store GenSets in a code-table, which is possible by adding a column for storing the code for n. There exist several codes for numbers (e.g., arithmetic codes). We choose a simple one, which requires the length $-\log_2(1/P)$ bits where P is the number of possible candidates, and n is bounded by the minimum size of transactions in $\mathcal{D}$. Assuming this code, we set $P = |\mathcal{I}|$, the worst length of such representation.

The Cover with GenSets For $d = (B, n) \in D$, if $n = 0$, d is an itemset B. We focus on how to define the cover with GenSets when $n > 0$.

Example 8. Let $d = (B, n) = (79, 1)$ and $t_3 = 12789$. We first cover a part of t_3 by the itemset B and get $128 = t_3 \setminus B$. Additionally we *rewrite* a symbol $\star$ $(n = 1)$ for $\{1, 2, 8\}$. An item 8 is chosen and $code_{ST}(8)$ is used for this choice because the code of 8 in ST is longer than that of 1 and 2.

Please see Fig. 4 as well for the cover of t_3 using a GenSet $d = (79, 1) \in D$. As the symbol $\star$ subsumes any items in $\mathcal{I}$ we need to evaluate such subsumptions. To implement it, we assume that we would like to find general, common features which characterize many transactions by excluding non-frequent items as long

[9] For three sets $X, Y, Z \in 2^{\mathcal{I}}$ we want to compute $n = \min(|X| - |XYZ|, |Y| - |XYZ|, |Z| - |XYZ|)$, where $XYZ = X \cap Y \cap Z$, and we see that this value n can be computed by the counting and set operations.

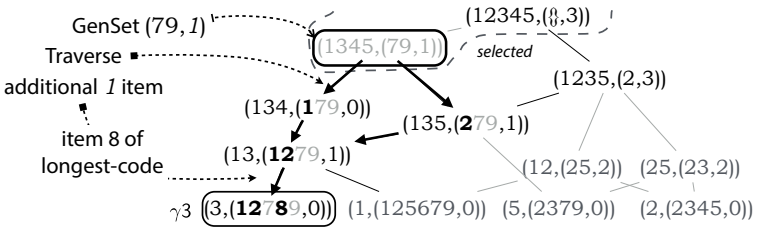


Fig. 4: For $t_3 = 12789$ with $CT = \{(79, 1)\}$, where $(79, 1)$ is a selected GenSet. An item can be covered by the GenSet $(79, 1)$ and we need to find it except 79.

as possible. We implement the subsumption of $\star$ as *replacing* the symbol $\star$ with $i \in \mathcal{I}$ by finding the item i having the longest $code_{ST}(i)$. Instead of giving a formal description, we give sketches of our idea below.

Sketch 1 We prepare a type SUB and construct an edit script as (SUB, i, c_i). As we have two types (DEL and SUB), $L(code_{\mathrm{OP}}(t))$ in Equation 3 is used. The function Encode$(\cdot)$ should be revised with traversing of an item i having the longest code. For example in Fig.4, $cover_{\mathrm{L}}(\{(79, 1)\}, 12789)$ now should work like $(\{(79, 1)\}, \{(\mathrm{DEL}, 1, c_1), (\mathrm{DEL}, 2, c_2), (\mathrm{SUB}, 8, c_8)\})$ after traversing $i = 8$.

Sketch 2 For $d = (B, n)$, if $n > 0$ (i.e., the code of n shows non-zero value), we use a code $code_{ST}(\{i\})$ for representing the item i of the longest code in ST without edit scripts. This is done by extending the function $cover_{\mathrm{L}}(\cdot, \cdot)$, as a function like $cover^{new}_{\mathrm{L}}(\{(79, 1)\}, 12789) = (\{(79, 1)\}, \{(\mathrm{DEL}, 1, c_1), (\mathrm{DEL}, 2, c_2)\}, \{8\})$, for example, where $\{8\}$ can be immediately evaluated by ST computed from $\mathcal{D}$.

In dealing with pattern concept lattices, many ways are applicable to fulfill the loss-less property. Because both sketches are based on our edit-tables for itemsets in Section 3 and they adopt prefix codes, they achieve the loss-less property if they have codes for items subsumed by $\star$. As they have such codes in fact, the loss-less property holds for GenSets. In experiments we use Sketch 2.

Corollary 1 (The Loss-Less Property of GenSets) The loss-less property immediately holds for GenSets from discussions above.

We stress that lattices are compact representations of objects and their features, and they help us to prepare edit operations (i.e., edit scripts) and the cover function. Furthermore, we can introduce more labels in subsumptions of $\star$ for representing various information to take into account valuable side information in pattern mining. In addition, we expect that GenSets can be used not only for the symbol $\star$ but also for representing hierarchy by stacking GenSets as trees.

Table 2: Benchmark datasets represented in the form of contexts (G, M, I). Recall that $\mathcal{CF}$ is the set of all closed itemsets with $\theta = 1$.

	objects $\lvert G\rvert$	attributes $\lvert M\rvert$	$\lvert I\rvert/(\lvert G\rvert\lvert M\rvert)$	$\lvert\mathcal{CF}\rvert$
IRIS	150	19	0.263	163
BREAST	699	16	0.624	641
PIMA	768	38	0.237	3,204
WINE	178	68	0.206	14,555

5 Experiments

From the UCI Machine Learning repository [10], we use IRIS[10], BREAST[11], PIMA[12], and WINE[13], which are also used in the KRIMP algorithm [18] as benchmark datasets. All of those datasets are available from the distribution package[14] of the KRIMP algorithm. The statistics of those datasets are summarized in Table 2.

All of our experiments were made on a machine of Mac OS X 10.10.2 with 2×2.26 GHz Quad-Core Xeon processors on 64GB main memory. All codes were written in Python 2.7.9. Our algorithm was the same as Algorithm 3 in the paper [18], which is an algorithm iteratively and greedy updating a code-table, except the part of the MDL evaluation. In our framework we replaced it with our implementation of Equations 3 and 4 based on lattices and edit operations. We did not discuss the computational complexity of our framework seriously. Rather than discussing solely run-times of experiments, we saw the followings.

- Comparison of two models: Results from two-models become the same by the equivalency of them (Proposition 1) for itemsets. We then investigate how much computational time our new model increases.
- We analyze the result with GenSets to check the application possibility of our lattice-based model for other patterns.

Let CT be the obtained code-table after applying the KRIMP algorithm. For a database $\mathcal{D}$ we can compute both $L(\mathcal{D}, ST)$ and $L(\mathcal{D}, CT)$. We then measured the *ratio* $L(\mathcal{D}, CT)/L(\mathcal{D}, ST)$ (used in [18]), which can be regarded as a kind of the degree of compression achieved by CT, denoted **Ratio** in results. In addition we also measured the size $\lvert CT^{-}\rvert$ because it indicates how many concepts are chosen out of the whole set of enumerated patterns. Furthermore, in experiments with GenSets, we measured the *Jaccard index*: $J(X, Y) \equiv \lvert X \cap Y\rvert/\lvert X \cup Y\rvert$ for two sets X and Y, between a set of identifiers of closed itemsets $\mathcal{CF}$ and that of GenSets, to see the difference between results by itemsets and those by GenSets. Note that in our setting, the set of pattern concepts by GenSets can be regarded as a set of closed itemsets if we ignore additional numbers n in $(B, n) \in D$.

[10] https://archive.ics.uci.edu/ml/datasets/Iris
[11] https://archive.ics.uci.edu/ml/datasets/Breast+Cancer+Wisconsin+(Diagnostic)
[12] https://archive.ics.uci.edu/ml/datasets/Pima+Indians+Diabetes
[13] https://archive.ics.uci.edu/ml/datasets/Wine
[14] http://www.patternsthatmatter.org/software.php, confirmed Feb. 2015.

Table 3: Results for itemsets by code-tables (the KRIMP algorithm) and two-tables (concepts and edit-tables in our framework).

				Krimp		**I-Edit**	
	$\lvert\mathcal{CF}\rvert$	$\lvert CT^{-}\rvert$	**Ratio**	Pre-proc. [s]	Run-time [s]	Pre-proc. [s]	Run-time [s]
IRIS	163	**13**	0.456	0.025	0.234	0.021	0.520
BREAST	641	**29**	0.181	0.580	6.263	0.573	26.794
PIMA	3,204	**66**	0.356	18.202	47.525	19.107	*373.63*
WINE	14,555	**72**	0.757	442.686	393.539	444.842	*1354.570*

In results, we labeled by **Krimp** to show results by the KRIMP algorithm, by **I-Edit** to show results by our framework using lattices and edit operations, and by **Gen-Edit** for results by GenSets using pattern concept lattices.

5.1 Results and Discussions for Itemsets

We show in Table 3 results of experiments for itemsets based on the existing method **Krimp** and our framework **I-Edit**, where we focus on run-times only. Note that we, of course, confirmed that results are equivalent.

With respect to the time for pre-processing, there existed no large differences. Concerning the time for the iterative updating process of models, roughly speaking our new model is at most 10 times slower than code-tables. Because both models are not implemented by sophisticated manners, we conjecture that our implementation cannot exploit pre-processed information on lattices well. That is, computing edit-tables using lattices is time-consuming and not sophisticated. Because the part of edit operations apparently depends on implementations of lattices and data structures for concepts, we expect that some advanced data structures help us to overcome this bottleneck.

5.2 Results and Discussions for GenSets

We conducted the same experiments using the same datasets with GenSets (in Section 4.1) and our MDL evaluation (in Section 4.2). Results are summarized in Table 4. We can see that the numbers of selected concepts via GenSets (i.e., 12, 24, 62, and 40) are *lower* than those via itemsets (i.e., 13, 29, 66, and 72), which mean that obtained models via GenSets of databases are *more compact* than those via itemsets. The same trend can be confirmed in the **Ratio** column.

The run-times of experiments have become faster than those in Table 3. Because the numbers of concepts (itemsets) and pattern concepts (GenSets) are the same now, these results should arise from the differences of the cover and the MDL evaluation. In addition the contents of the selected patterns should be affected. We can see the difference of the contents of the selection by checking the Jaccard index. As we can see that, our GenSets formulation surely generates *completely* different results compared with the results for itemsets.

Table 4: Results by our lattice-based framework for GenSets.

				Gen-Edit		
	$\lvert\mathcal{CF}\rvert$	$\lvert CT^{-}\rvert$	**Ratio**	Pre-proc. [s]	Run-time [s]	Jaccard Index
IRIS	163	**12**	0.428	0.009	0.550	0.785
BREAST	641	**24**	0.168	0.351	20.666	0.293
PIMA	3,204	**62**	0.342	18.92	285.676	0.123
WINE	14,555	**40**	0.686	415.376	454.631	0.131

Furthermore, the existence of stars $\star$ in GenSets affect the results. For GenSets (in Section 4.2), we aggressively cover items having longer codes in ST if the symbol $\star$ appears. As less frequent itemsets have longer codes in the table ST and such items are covered by $\star$ in advance, concepts stored in our model have relatively high frequency in the database, and the results get more compact.

Concluding Remarks on Experiments, and Future Directions We conducted the MDL-based pattern summarization for selected benchmark datasets by using code-tables and our model (two tables). We also applied our model for GenSets to which we can apply the same framework. At least we confirmed that our model obtained more compact results using GenSets (checked by **Ratio**), and the contents are also different (done by Jaccard index).

With respect to the *quality of the choice* of concepts via the two-part MDL principle it is always open to discuss. To discuss the quality, the previous work [18] developed the classifier using code-tables, tested it for labeled data, and discussed the quality of their choice. Therefore following their strategy and developing a classifier using edit operations and lattices are our future work to consider the pros and cons of our model and the quality of the choice.

Further discussion on the relation of run-times for edit-tables and properties of lattices (e.g., the density of contexts), is also important. Because the order of checking the set $\mathcal{CF}$ is optional, developing and analyzing further heuristic based on lattices is also important. We expect that several terminologies (not used in this paper) and methods for lattices would be helpful for this direction.

6 Conclusions and Future Work

In this paper we propose a new framework for pattern summarization to conquer the redundancy problem for various patterns by combining lattices and edit operations, which are evaluated by the two-part MDL principle. Our experiments show that our model successfully extends an existing model towards more general patterns. As an example we confirm that our model is applicable to GenSets, defined by pattern structures as a richer representation of itemsets. For closed itemsets, our model achieves the same results as the existing model (Proposition 1). Results of **Ratio** in experiments, GenSets are useful to obtain more compact representations of database compared with an existing model.

In our future work, we plan to develop a classifier based on the MDL principle by following the existing work for discussing the quality of the selection. We also investigate encoding methods and heuristic for the cover towards various patterns together with side information. Theoretical analysis of the two-part MDL evaluation via lattices is also our important future research direction.

Acknowledgements. The authors are grateful to the anonymous reviewers for valuable comments. This study was partially supported by Grant-in-Aid for JSPS Fellows (26-4555) and JSPS KAKENHI Grant Number 26280085.

References

1. Aggarwal, C.C., Han, J. (eds.): Frequent pattern mining. Springer International Publishing (2014)
2. Agrawal, R., Srikant, R.: Fast algorithms for mining association rules in large databases. In: Proc. of the 20th VLDB. pp. 487–499 (1994)
3. Buzmakov, A., Egho, E., Jay, N., Kuznetsov, S.O., Napoli, A., Raïssi, C.: The representation of sequential patterns and their projections within formal concept analysis. In: Workshop Notes for LML (ECML/PKDD2013) (2013)
4. Ganter, B., Kuznetsov, S.O.: Pattern structures and their projections. In: Proc. of the 9th ICCS. pp. 129–142 (2001)
5. Ganter, B., Wille, R.: Formal concept analysis - mathematical foundations. Springer (1999)
6. Geerts, F., Goethals, B., Mielikäinen, T.: Tiling databases. In: Proc. of the 7th DS. pp. 278–289 (2004)
7. Grünwald, P.D.: The minimum description length principle. The MIT Press (2007)
8. Koutra, D., Kang, U., Vreeken, J., Faloutsos, C.: VOG: Summarizing and understanding large graphs. In: Proc. of the 22nd SDM. pp. 91–99 (2014)
9. Li, M., Vitnyi, P.M.: An introduction to kolmogorov complexity and its applications. Springer Publishing Company, Incorporated, 3 edn. (2008)
10. Lichman, M.: UCI machine learning repository (2013), http://archive.ics.uci.edu/ml
11. Otaki, K., Ikeda, M., Yamamoto, A.: Pattern structures for understanding episode patterns. In: Proc. of the 11th CLA. pp. 47–58 (2014)
12. Pasquier, N., Bastide, Y., Taouil, R., Lakhal, L.: Discovering frequent closed itemsets for association rules. In: Prof. of the 7th ICDT. pp. 398–416 (1999)
13. Pei, J., Han, J., Mao, R.: CLOSET: An efficient algorithm for mining frequent closed itemsets. In: Proc. of the DMKD2000. pp. 21–30 (2000)
14. Smets, K., Vreeken, J.: Slim: directly mining descriptive patterns. In: Proceedings of the 20th SDM. pp. 236–247 (2012)
15. Tatti, N., Vreeken, J.: Discovering descriptive tile trees. In: Proc. of the ECML/PKDD 2012. pp. 24–28 (2012)
16. Tatti, N., Vreeken, J.: The long and the short of it: Summarising event sequences with serial episodes. In: Proc. of the 18th KDD. pp. 462–470 (2012)
17. Uno, T., Asai, T., Uchida, Y., Arimura, H.: LCM: An efficient algorithm for enumerating frequent closed item sets. In: Proc. of the FIMI'03 (2003)
18. Vreeken, J., van Leeuwen, M., Siebes, A.: Krimp: Mining itemsets that compress. Data Mining and Knowledge Discovery 23(1), 169–214 (2011)