Pattern Structures for Understanding Episode Patterns

Keisuke Otaki*, Ikeda Madori and Akihiro Yamamoto

Department of Intelligence Science and Technology
Graduate School of Informatics, Kyoto University, Japan.
{ootaki,m.ikeda}@iip.ist.i.kyoto-u.ac.jp, akihiro@i.kyoto-u.ac.jp

Abstract. We investigate an application of pattern structures for understanding episodes, which are labeled directed acyclic graphs representing event transitions. Since typical episode mining algorithms generate a huge number of similar episodes, we need to summarize them or to obtain compact representations of them for applying the outcome of mining to various problems. Though such problems have been well-studied for itemsets, summarization of episodes is still understudied. For a class called diamond episodes, we first provide a pattern structure based on hierarchy of events to obtain small groups of episodes in the form of pattern concepts and lattice structures. To find a summary via pattern concepts, we design an utility function for scoring concepts. After ranking concepts using some function and lattice structures, we try to sample a set of pattern concepts of high scores as a summary of episodes. We report our experimental results of our patten structure, and a ranking result of our simple utility function. Last we discuss pattern concept lattices and their applications for summarization problems.

Keywords: formal concept analysis, pattern structure, episode pattern, pattern summarization

1 Introduction

Knowledge Discovery from binary databases is a fundamental problem setting, where binary databases represent that some objects have some features by their 1 entries. Because such a situation can be seen in many practical problems, both theoretical and practical aspects of the problems have been studied.

On a mathematical viewpoint, Formal Concept Analysis (FCA) [5] has been studied as a model of analyzing such binary databases. We deal with a *context* $K = (O, A, I)$ consisting of 1) a set O of objects, 2) a set A of attributes, and 3) a binary relation $I \subseteq O \times A$ representing that an i-th object has a j-th attribute. FCA adopts two functions f and g for analyzing O and A; f receives a set of objects and returns a set of attributes which are commonly possessed by given objects, and g receives a set of attributes and returns a set of objects which

* Keisuke Otaki is supported as a JSPS research fellow (DC2, 26·4555).

have commonly the input attributes. For $X \subseteq O$ and $Y \subseteq A$, a tuple (X, Y) is called a *concept* if $f(X) = Y$ and $X = g(Y)$. Computing the set of all concepts is a fundamental but important task in FCA, which help us to analyze binary databases. On a practical viewpoint, it is well-known that formal concepts are related to closed itemsets studied in frequent itemset mining [13], which are also known as compact representations of itemsets.

To deal with non-binary data in an FCA manner, pattern structures [4] have been studied. A key idea is generalizing both the set intersection $\cap$ and the subset relation $\subseteq$, which are used in two functions f and g in FCA. The set intersection $\cap$ is replaced with a meet operator $\sqcap$ that extracts *common substructures* of two objects. The subset relation $\subseteq$ is also replaced with a partial order $\sqsubseteq$ induced by $\sqcap$, where $\sqsubseteq$ represents some *embedding* from an object into another. We now assume that they would help us to understand complex data.

In this paper, we investigate pattern structures and their applications for understanding patterns, motivated by a requirement of summarization techniques because a large numbers of patterns is always generated by some mining algorithms. As an example in this paper, we deal with some classes of *episode patterns*, which represent event transitions in the form of labeled graphs. From such patterns, we can compute lattice structures based on pattern structures (Section 3). Since such lattices represent mutual relations among patterns and several small clusters as pattern concepts, analyzing them would be helpful to obtain a small set of pattern concepts. We regard a subset of all concepts as a summary of features often used in describing patterns, and develop a way of obtaining a small set of concepts as a summary. When we construct descriptions of objects, we also introduce the wildcard $\star$ as a special symbol representing all events to take into account some hierarchy of labels based on our knowledge. It would be a strong merit of pattern structures for summarization in which similar patterns could be merged into some descriptions with the wildcard $\star$. After providing pattern structures, we provide preliminary experimental results (Section 4) and discuss it on the viewpoint of summarization by giving a utility function for ranking pattern concepts (Section 5).

2 Formal Concept Analysis and Episode Mining

FCA and Pattern Structures We adopt the standard notations of FCA from [5] and pattern structures from [4], respectively. Here we refer the notations of FCA which we have already used in Section 1. For a context $K = (O, A, I)$, $X \subseteq O$ and $Y \subseteq A$, two functions f and g in FCA are formally defined by $f(X) = \{a \in A \mid (o, a) \in I \text{ for all } o \in X\}$ and $g(Y) = \{o \in O \mid (o, a) \in I \text{ for all } a \in Y\}$, respectively. Recall that a pair (X, Y) is a *(formal) concept* if $f(X) = Y$ and $g(Y) = X$. Two operators $f \circ g(\cdot)$ and $g \circ f(\cdot)$ are *closure operators* on 2^O and 2^A, respectively. Note that a concept (X, Y) is in the form either $(g(f(X)), f(X))$ or $(g(Y), f(g(Y)))$. For two concepts (X_1, Y_1) and (X_2, Y_2), the partial order $\leq$ is introduced by $X_1 \subseteq X_2$ ($\Leftrightarrow Y_2 \subseteq Y_1$).

An important aspect of pattern structures is generalization of two operations $\cap$ and $\subseteq$ used in f and g. They are characterized by *meet semi-lattices*: A *meet semi-lattice* $(D, \sqcap)$ of a set D and a meet operator $\sqcap$ is an algebraic structure satisfying: 1) Associativity; $x \sqcap (y \sqcap z) = (x \sqcap y) \sqcap z$ for $x, y, z \in D$, 2) Commutativity; $x \sqcap y = y \sqcap x$ for $x, y \in D$, and 3) Idempotency; $x \sqcap x = x$ for $x \in D$. Elements in D are called *descriptions*. A partial order $\sqsubseteq$ is induced by $\sqcap$ as

$$x \sqsubseteq y \text{ whenever } x \sqcap y = x \text{ for two elements } x, y \in D.$$

Example 1 (A meet semi-lattice with sets). Let $D = 2^{\mathbb{N}}$. For two sets of integers $X = \{1, 2\}$ and $Y = \{1, 2, 3, 4\}$, it holds that $X \cap Y = X$, which induces $X \subseteq Y$.

Example 2 (A meet semi-lattice for closed intervals [8]*).* Let D be a set of all closed intervals $[a, b]$ with integers $a, b \in \mathbb{N}$. They define $\sqcap$ of two closed intervals by keeping its convexity, that is, $[a_1, b_1] \sqcap [a_2, b_2] = [\min(a_1, a_2), \max(b_1, b_2)]$.

By generalizing a meet semi-lattice $(2^A, \cap)$ used in FCA, a pattern structure $\mathbb{P}$ is defined by a triple $(O, (D, \sqcap), \delta)$, where O is the set of objects, $(D, \sqcap)$ is a meet semi-lattice of *descriptions*, and $\delta : O \to D$ is a mapping of giving a description for each object. For analyzing pattern structures, we obtain the following Galois connection $\{(\cdot)^{\square}, (\cdot)^{\diamond}\}$ corresponding to f and g in FCA:

$$A^{\square} = \sqcap_{o \in A} \delta(o) \text{ for } A \subseteq O, \tag{1}$$

$$d^{\diamond} = \{o \in O \mid d \sqsubseteq \delta(o)\} \text{ for } d \in D. \tag{2}$$

Pattern concepts based on $\mathbb{P}$ are defined by Equations (1) and (2):

Definition 1 (Pattern concepts) A *pattern concept* of $\mathbb{P}$ is a pair (A, d) of a set $A \subseteq O$ and a pattern $d \in D$ satisfying $A^{\square} = d$ and $d^{\diamond} = A$. Two pattern concepts are partially ordered by $(A_1, d_1) \preceq (A_2, d_2)$ by $A_1 \subseteq A_2 \Leftrightarrow d_2 \sqsubseteq d_1$.

Note that by its partial order, the set of all pattern concepts forms a lattice structure. We denote the set of all pattern concepts by $\mathfrak{P}(\mathbb{P})$. To obtain $\mathfrak{P}(\mathbb{P})$, we need to compute two functions $(\cdot)^{\square}$ and $(\cdot)^{\diamond}$. For example, we can adopt the AddIntent proposed in [12] and used in [8].

Episode Mining We briefly review episode mining based on [7]. Let $\mathcal{E} = \{1 \ldots, m\} \subseteq \mathbb{N}$ be the set of *events*. We call a set $S \subseteq \mathcal{E}$ of events an *event set*. An input of episode mining is a long sequence of event sets; an *input event sequence* $\mathcal{S}$ on $\mathcal{E}$ is a finite sequence $\langle S_1, \ldots, S_n \rangle \in (2^{\mathcal{E}})^*$, where each set $S_i \subseteq \mathcal{E}$ is the i-th event set. For $\mathcal{S}$ of length n, we assume that $S_i = \emptyset$ if $i < 0$ or $i > n$.

Episodes are *labeled directed graphs* (DAGs). An episode G is a triple (V, E, λ), where V is the set of vertices, E is the set of directed edges, and λ is the labeling function from V and E to the set of labels, that is, $\mathcal{E}$. Several classes of episodes have been studied since episode mining is firstly introduced by Mannila *et. al.* [11]. We follow subclasses of episodes studied by Katoh *et al.* [7]. An

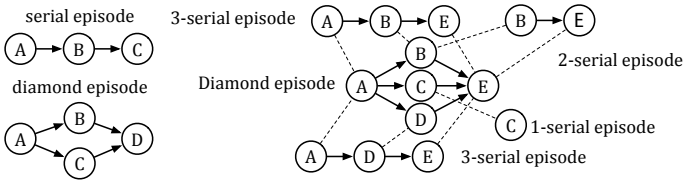

Fig. 1. An example of episode studied in episode mining in [11] and [7].

example of episodes is illustrated in Figure 1. In designing pattern mining algorithms, we need 1) a search space of patterns and a partial order for enumerating patterns, and 2) interestingness measure to evaluate them. For episode mining, we often adopt *occurrences* of episodes defined with *windows*.

Definition 1 (Windows). For a sequence $\mathcal{S} = \langle S_1, \ldots, S_n \rangle$, an *window* W of $\mathcal{S}$ is a contiguous subsequence $\langle S_i, \cdots, S_{i+w-1} \rangle$ of length n, called *width*, for some index i $(-w+1 \leq i \leq n)$ of $\mathcal{S}$ and a positive integer $w \geq 0$.

Definition 2 (Embedding of Episodes). Let $G = (V, E, \lambda)$ be an episode, and $W = \langle S_1, \ldots, S_w \rangle$ be a window of width w. We say that G *occurs* in W if there exists a mapping $h : V \to \{1, \ldots, w\}$ satisfying 1) for all $v \in V$, $h(v) \in S_{h(x)}$, and 2) for all $(u, v) \in E$ with $u \neq v$, it holds that $h(u) < h(v)$. The map h is called an *embedding* of G into W, and it is denoted by $G \preceq W$.

For an input event sequence $\mathcal{S}$ and a episode G, we say that G occurs at position i of $\mathcal{S}$ if $G \preceq W_i$, where $W_i = \langle S_i, \ldots, S_{i+w-1} \rangle$ is the i-th window of width w in $\mathcal{S}$. We then call the index i an *occurrence* of G in $\mathcal{S}$. The *domain* of the occurrences is given by $\mathbf{W}_{\mathcal{S},w} = \{i \mid -w+1 \leq i \leq n\}$. In addition, $\mathbf{W}_{\mathcal{S},w}(G)$ is the *occurrence window list* of an episode G, defined by $\{-w+1 \leq i \leq n \mid G \preceq W_i\}$. Then we can define an interestingness measure *frequency* of episodes.

Definition 3 (Frequency of Episodes). The *frequency* of an episode G in $\mathcal{S}$ and w, denoted by $\mathit{freq}_{\mathcal{S},w}(G)$, is defined by the number of windows of width w containing G. That is, $\mathit{freq}_{\mathcal{S},w}(G) = |\mathbf{W}_{\mathcal{S},w}(G)|$. For a threshold $\theta \geq 1$, a width w and an input event sequence $\mathcal{S}$, if $\mathit{freq}_{\mathcal{S},w}(G) \geq \theta$, G is called *θ-frequent* on $\mathcal{S}$.

The frequent episode mining problem is defined as follows: Let $\mathcal{P}$ be a class of episodes. Given an input event sequence $\mathcal{S}$, a width $w \geq 1$, and a frequency threshold $\theta \geq 1$, the problem is to find all θ-frequent episodes G belonging to the class $\mathcal{P}$. The simplest strategy of finding all θ-frequent episodes is traversing $\mathcal{P}$ by using the anti-monotonicity of the frequency count $\mathit{freq}(\cdot)$. For details, we would like to refer to both [7] and [11].

For our examples of classes, we introduce m-serial episodes and diamond episodes. An *m-serial episode* over $\mathcal{E}$ is a sequence of events in the form of $a_1 \mapsto a_2 \mapsto \cdots \mapsto a_m$. A *diamond episode* over $\mathcal{E}$ is either 1) a 1-serial episode $e \in \mathcal{E}$ or 2) a *proper diamond episode* represented by a triple $Q = \langle a, X, b \rangle \in \mathcal{E} \times 2^{\mathcal{E}} \times \mathcal{E}$, where a, b are events and $X \subseteq \mathcal{E}$ is an event set occurring after a and before

b. For short, we write a diamond episode as $a \mapsto X \mapsto b$. On the one hand definitions of episodes by graphs are much general, on the another hand classes of episode patterns are often restricted.

Example 3 (Episodes). In Figure 1, we show some serial episodes; $A \mapsto B \mapsto E$, $A \mapsto D \mapsto E$, $B \mapsto E$, and C on the set of events $\mathcal{E} = \{A, B, C, D, E\}$. All of them are included in a diamond episode $A \mapsto \{B, C, D\} \mapsto E$.

We explain a merit of introducing pattern structures for summarization of structured patterns. As we mentioned above, a common strategy adopted in pattern mining is traversing the space $\mathcal{P}$ in a breadth-first manner with checking some interestingness measure. When generating next candidates of frequent patterns, algorithms always check a *parent-child* relation between two patterns. This order is essential for pattern mining and we thus conjecture that this parent-child relation used in pattern mining can be naturally adopted in constructing a pattern structure for analyzing patterns only by introducing a similarity operation $\sqcap$. After constructing a lattice, it would be helpful to analyze a set of all patterns using it because they represent all patterns compactly.

A crucial problem of pattern structures is the computational complexity concerning both $\sqcap$ and $\sqsubseteq$. Our idea is to adopt trees of height 1 (also called *stars* in Graph Theory). That is, we here assume that trees are expressive enough to represent features of episodes. Our idea is similar that used in designing graph kernels [14][1] and that is inspired by previous studies on pattern structures [2, 4].

3 Diamond Episode Pattern Structures

In the following, we focus on diamond episodes as our objects, and *trees of height* 1 as our descriptions. They have two special vertices; the source and the sink. They can be regarded as important features for representing event transitions. We generate rooted labeled trees from them by putting the node in the root of a tree, and regarding neighbors as children of it. Since heights of all trees here are 1, we can represent them by tuples without using explicit graph notations.

Definition 4 (Rooted Trees of Height 1). Let $(\mathcal{E}, \sqcap_{\mathcal{E}})$ be a meet semi-lattice of event labels. A rooted labeled tree of height 1 is represented by a tuple [2] $(e, C) \in \mathcal{E} \times 2^{\mathcal{E}}$. We represent the set of all rooted labeled trees of height 1 by $\mathfrak{T}$.

Note that in $(\mathcal{E}, \sqcap_{\mathcal{E}})$, we assume that $\sqcap_{\mathcal{E}}$ compares labels based on our background knowledge. We need to take care that this meet semi-lattice $(\mathcal{E}, \sqcap_{\mathcal{E}})$ is independent and different from a meet semi-lattice $\underline{\mathrm{D}}$ of descriptions of a pattern structure $\mathfrak{P}$. This operation $\sqcap_{\mathcal{E}}$ is also adopted when defining an embedding of trees of height 1, that is, a partial order between trees defined as follows.

[1] It intuitively generates a sequence of graphs by relabeling all vertices of a graph. One focus on a label of a vertex $v \in V(G)$ and sees labels $LN_G(v)$ of its neighbors $N_G(v)$. For a tuple $(l_v, LN_G(v))$ for all vertices $v \in V(G)$, we sort all labels lexicographically, and we assign a new label according to its representation. Details are seen in [14].

[2] On the viewpoint of graphs, this tuple (e, C) should represent a graph $G = (V, E, \lambda)$ of $V = \{0, 1, \ldots, |C|\}$, $E = \{(0, i) \mid 1 \leq i \leq |C|\}$, $\lambda(0) = e$, $\{\lambda(i) \mid 1 \leq i \leq |C|\} = C$.

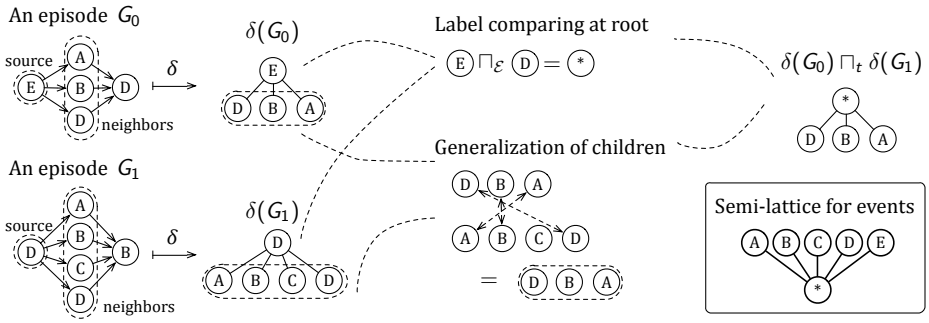

Fig. 2. An example of computations $\sqcap$ of two trees of height 1.

Definition 5 (Partial Order on Trees). A tree $t_1 = (e_1, C_1)$ is a *generalized subtree* of $t_2 = (e_2, C_2)$, denoted by $t_1 \sqsubseteq_T t_2$, iff $e_1 \sqsubseteq_{\mathcal{E}} e_2$ and there exists an injection mapping $\phi : C_1 \to C_2$ satisfying for all $v \in C_1$, there exists $\phi(v) \in C_2$ satisfying $v \sqsubseteq_{\mathcal{E}} \phi(v)$, where $\sqsubseteq_{\mathcal{E}}$ is the induced partial order by $\sqcap_{\mathcal{E}}$.

For defining a similarity operator $\sqcap_T$ between trees, this partial order $\sqsubseteq_T$ is helpful because $\sqcap_T$ is closely related to $\sqsubseteq_T$ in our scenario. Since all trees here are height 1, this computation is easy to describe; For labels of root nodes, a similarity operator is immediately given by using $\sqcap_{\mathcal{E}}$. For their children, it is implemented by using an idea of *least general generalization* (LGG), which is used in Inductive Logic Programming [10], of two sets of labels. A practical implementation of LGG depends on whether or not sets are *multisets*, but it is computationally tractable. An example is seen in Figure 2.

We give formal definitions of δ and D. For a graph $G = (V, E, \lambda)$, we denote the neighbors of $v \in V$ by $N_G(v)$. For some proper diamond episode pattern G, the source vertex $s \in V$ and the sink vertex $t \in V$, computed trees of height 1 corresponding s and t are defined as $T_s = (\{s\} \cup N_G(s), \{(s, u) \mid u \in N_G(s)\}, \lambda)$, and $T_t = (\{t\} \cup N_G(t), \{(u, t) \mid u \in N_G(t)\}, \lambda)$, respectively. By using those trees, $\delta(\cdot)$ can be defined according to vertices s and t: If we see both T_s and T_t, $\delta(G) = (T_s, T_t)$ and then $\sqcap_T$ is adopted element-wise, and D is defined by $\mathfrak{T} \times \mathfrak{T}$. If we focus on either s or t, $\delta(G) = T_s$ or T_v, and we can use $\sqcap_T$ directly by assuming $D = \mathfrak{I}$.

Last we explain relations between our pattern structures and previous studies shortly. This partial order $\sqsubseteq_T$ is inspired from a generalized subgraph isomorphism [4] and a pattern structure for analyzing sequences [2]. We here give another description of similarity operators based on definition used in [4, 9].

Definition 6 (Similarity Operation $\sqcap$ based on [9]). The similarity operation $\sqcap$ is defined by the set of all maximal common subtrees based on the generalized subtree isomorphism $\sqsubseteq_T$; For two trees s_1 and s_2 in $\mathfrak{T}$,

$$s_1 \sqcap s_2 \equiv \{u \mid u \sqsubseteq_T s_1, s_2, \text{ and } \forall u' \sqsubseteq_T s_1, s_2 \text{ satisfying } u \not\sqsubseteq_T u'\}.$$

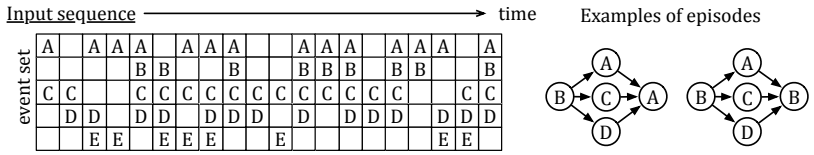

Fig. 3. An input $\mathcal{S}$ and two diamond episodes mined from $\mathcal{S}$ as examples.

Table 1. Numbers of proper diamond episodes and pattern concepts for $w \in \{3, 4, 5\}$ and $M \in \{100, 200, 300, 400, 500, 600, 700\}$. In the table below, DE and PDE means Diamond Episodes and Proper Diamond Episodes, respectively.

			M and # of pattern concepts						
Window width w	# of DE	# of PDE	100	200	300	400	500	600	700
3	729	569	87	137	178	204	247	–	–
4	927	767	74	136	179	225	281	316	336
5	935	775	71	137	187	272	290	313	342

Note that we can regard that our operator $\sqcap_T$ is a special case of the similarity operation $\sqcap$ above. On the viewpoint of pattern structures, our trees of height 1 can be regarded as an example of projections from graphs into trees, studied in [4, 9], such as both k-chains (paths on graphs of length k) and k-cycles.

4 Experiments and Discussion for Diamond Episodes

Data and Experiments We gathered data from MLB baseball logs, where a system records all pitching and plays for all games in a season. We used what types of balls are used in pitching, which can be represented by histograms per batter. For a randomly selected game, we generated an input event sequence of episode mining by transforming each histogram to a set of types of balls used types of balls[3]. In forming $(\mathcal{E}, \sqcap_{\mathcal{E}})$, we let $\mathcal{E}$ be the set of types of balls, and define $\sqcap_{\mathcal{E}}$ naturally (See Example in Fig. 2). For this $\mathcal{S}$, we applied a diamond episode mining algorithm proposed by [7] and obtain a set of diamond episodes. The algorithm have two parameters; the window size w and the frequency threshold θ. We always set $\theta = 1$ and varied $w \in \{3, 4, 5\}$. After generating a set $\mathcal{G}$ of frequent proper diamond episodes, we sampled $M \in \{100, 200, \ldots, 700\}$ episodes from $\mathcal{G}$ as a subset O of $\mathcal{G}$ (that is, satisfying $|O| = M$ and $O \subseteq \mathcal{G}$). We used O as a set of objects in our pattern structure $\mathbb{P}$. From it we computed all pattern concepts $\mathfrak{P}(\mathbb{P})$ based on our discussions in Section 3. In this experiments we set $\delta(G) = T_s$ for a proper diamond episode G and its source vertex s.

[3] In baseball games, pitchers throw many kinds of balls such as fast balls, cut balls, curves, sinkers, etc. They are recorded together with its movements by MLB systems.

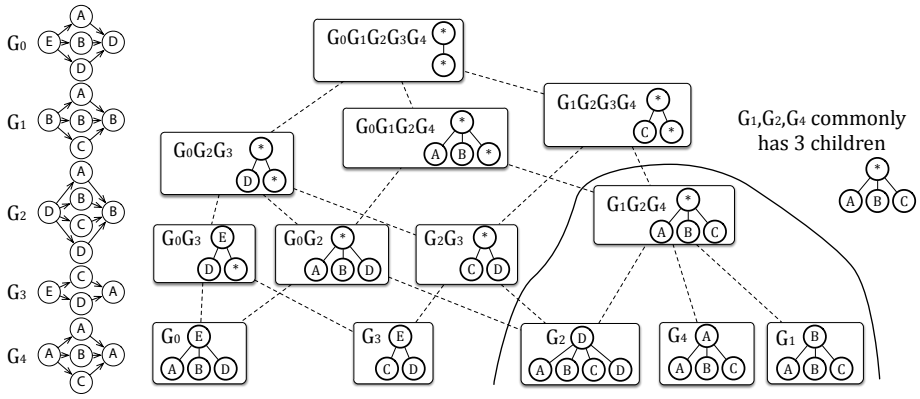

Fig. 4. A small example of a pattern structure lattice.

For experiments, we adopted an implementation by Katoh *et al.* [7] for mining diamond episodes, which is written in C++[4]. We implemented the Galois connection $\{(\cdot)^{\square}, (\cdot)^{\diamond}\}$ by the AddIntent [12] algorithm using Python[5].

Results and Discussion Table 1 shows the results of numbers of pattern concepts and those of proper diamond episodes for each set corresponding $w = 3, 4$, and 5, with varying $M \in \{100, 200, 300, 400, 500, 600, 700\}$. In Figure 4, we show a result of pattern concepts and a pattern concept lattice computed from only first 5 episodes for the case $w = 5$, as an example of our lattice structures.

Because we assume a semi-lattice $(\mathcal{E}, \sqcap_{\mathcal{E}})$ of events in episode patterns, we can obtain pattern concepts in which some vertices are represented by the wildcard $\star$. If we implement $\sqcap_t$ without the wildcard $\star$, we can only obtain a much smaller number of pattern concepts compared with our results including the wildcard $\star$. We thus conjecture that the wildcard $\star$ is useful to represent similar patterns in Figure 4. On such a viewpoint, pattern structures and pattern concepts help us to make some group of patterns, which are basically similar to clustering patterns. Here we do not discuss details of computational complexity of constructing pattern concept lattices, but the complexity basically depends on the number of concepts. Thus it would be interesting to investigate and compare several possible projections and computations of pattern concepts.

5 Pattern Summarization and Pattern Structures

In this section, we discuss pattern summarization based on pattern concept lattices. As we mentioned, closed itemsets [13] have been studied as compact

[4] We adopt GCC4.7 as our compiler including c++11 features (-STD=C++11). The code is compiled on a machine of Mac OS X 10.9 with two 2.26 GHz Quad-Core Intel Xeon Processors and 64GB main memory.

[5] We used Python 2.7 without any additional packages (that is, pure python).

representations of itemsets, and they are closely related to the closure operator $g \circ f$ in FCA with (O, A, I), where O is the set of transaction identifiers and A is the set of all items. The difficulty of closed patterns for complex data is there are no common definitions of closure operators, where we usually use the closeness with respect to the frequency. Here we assume that pattern concepts are helpful in the same correspondence between closed itemsets and concepts.

To obtain some compact representations, we need to decide *how to evaluate each pattern*. The problem here is how to deal with the wildcard $\star$ in descriptions. When we obtain a concept (X, Y) for $X \subseteq O, Y \subseteq A$, this concept (X, Y) corresponds to a rectangle on I, and there are no 0 entries in the sub-database $I' = \{(x, y) \in I \mid x \in X, y \in Y\}$ of I induced by (X, Y) because of its definitions. If (X, Y) is not a concept, a rectangle r by (X', Y') contains a few 0 entries in it. We denote the relative ratio of 1 entries in a rectangle r by (X', Y') as

$$\mathbf{r}_1(X', Y', I) = (1 - |\{(x, y) \notin I \mid x \in X', y \in Y'\}|) \left(|X'||Y'|\right)^{-1},$$

where $0 \leq \mathbf{r}_1(X', Y', I) \leq 1$ and $\mathbf{r}_1(X', Y', I) = 1$ if (X', Y') is a concept. These $\mathbf{r}_1(X, Y, I)$, $|X|$, and $|Y|$ are applicable for evaluating itemsets. If we only use the cardinality $|A|$ of a set A of objects, this equals to the support counts computed in *Iceberg concept lattices* [15]. For a concept (X, Y) of a context $K = (O, A, I)$, we compute the support count $supp(X, Y) = |g(Y)|/|O|$ and prune redundant concepts by using some threshold. For formalizing evaluations of patterns, such values are generalized by introducing a *utility function* $u : \mathcal{P} \to \mathbb{R}^+$. A typical and well-studied utility function is, of course, the *frequency count*, or the *area function* $area(\cdot)$ which evaluates the size of a rectangle (X, Y) [6].

Based on discussions above, if we can define a utility function $u(\cdot)$ for evaluating pattern concepts, a similar discussion for pattern concepts are possible; choosing a few number of pattern concepts and constructing summary of patterns with them. Of course, there are no simple way of giving such functions. We try to introduce a simple and straightforward utility function $u_P(\cdot)$ for pattern concepts as a first step of developing pattern summarization via pattern concept lattices. In this paper, we follow the idea used in tiling databases [6], where a key criterion is given by $area(\cdot)$. We consider how to compute the value which corresponds to the area in binary databases. To take into account the wildcard $\star$ used in descriptions, we define the following simple function. For $d \in D$, we let $s(d)$ and $n(d)$ be the numbers of non wildcard and all vertices in a description d, respectively. Note that if $s(d) = n(d)$, d contains no wildcard labels. By using these functions, we compute utility values as follows:

$$u_P(A, d) = |A| \cdot \log\left(1 + s(d)\right).$$

5.1 Experiments and Discussions

We compare results of ranking pattern concepts by 1) using only $|A|$ (similar to the Iceberg concept lattices), and 2) using $u_P(\cdot)$ as a utility function. From the list of pattern concepts generated in experiments of Section 4, we rank all

Table 2. Results of ranking pattern concepts from 750 episodes in $w = 5$.

Utility	Top-5 mutually distinct descriptions of pattern concepts
$\|A\|$	$(\star, \{\star\})$, $(2, \{\star\})$, $(0, \{\star\})$, $(3, \{\star\})$, $(1, \{\star\})$
$u_P(\cdot)$	$(\star, \{0, \star\})$, $(\star, \{0, 2, 3\})$, $(\star, \{0, 1, 2\})$, $(\star, \{0, 1, 3\})$, $(\star, \{1, 2, 3\})$

pattern concepts by using a utility function, and sort the list in an ascending order, and compare two lists. We remove patterns appearing commonly in both lists to highlight differences. We give our results in Table 2.

In the result with $u_P(\cdot)$, larger descriptions appear with higher utility values compared with those by $|A|$. We can see that by modifying terms concerning $\star$, results contain more informative nodes, which are labeled by non-wildcard labels. Here we implicitly assume that descriptions contains less $\star$ would be more useful for understanding data themselves. On this viewpoint, considering two terms $s(d)$ and $n(d)$ for description d would be interesting and useful way to design utility functions for pattern concepts. We conclude that the Iceberg lattice based support counts are less effective if descriptions admit the wildcard $\star$ for pattern summarization problems.

Not only the simple computation in $u_P(A, d)$ used above, also many alternatives could be applicable for ranking. Some probabilistic methods such as the minimum description length (MDL), information-theoretic criteria would be also helpful to analyze our study more clearly. Since pattern structures have no explicit representations of binary cross tables, the difficulty lies on how to deal with a meet semi-lattice $(D, \sqcap)$. For some pattern concept (A, d) and an object $o \in O$, we say that (A, d) *subsumes* o if and only if $d \sqsubseteq \delta(o)$. This subsumption relation would be simple and helpful to evaluate concepts, but they does not adopt any complex information concerning hierarchy of events, or distances between two descriptions. In fact in the experiments, we always assume that all events except $\star$ have the same weight and $\star$ is the minimum of all events. They could be important to take into account similarity measures of events for more developments of ranking methods of pattern concepts.

5.2 Related Work

There are several studies concerning our study. It is well-known that closed itemsets correspond to maximal bipartite cliques on bipartite graphs constructed from $K = (O, A, I)$. Similarly, we sometimes deal with so called *pseudo bipartite cliques* [16], where it holds that $\mathbf{r}_1(X', Y', I) \geq 1 - \varepsilon$ with a user-specified constance ε. Obviously, pseudo bipartite cliques correspond to rectangles containing a few 0. We can regard them as some summarization or approximation of closed itemsets or concepts. Intuitively, if we use some pseudo bipartite cliques as summarization, the value $\mathbf{r}_1(X, Y, I)$ can be considered in evaluating (X, Y). Pseudo bipartite cliques can be regarded as noisy tiles, which is an extension of tiles [6].

Another typical approach for summarization is clustering patterns [18, 1]. A main problem there is how to interpret clusters or centroids, where we need to de-

sign a similarity measure and a space in which we compute the similarity. On the viewpoint of probabilistic models, there is an analysis via the maximum entropy principle [3]. However they assume that entries in a database are independently sampled, and thus we cannot apply those techniques to our setting.

6 Toward Generalizations for Bipartite Episodes

In this paper we assume that our descriptions by trees of height 1 are rich enough to apply many classes of episode patterns. We here show how to apply our pattern structure for other types of episodes, called *bipartite episodes*, as an example. An episode $G = (V, E, \lambda)$ is a *a partial bipartite episode* if 1) $V = V_1 \cup V_2$ for mutually disjoint sets V_1 and V_2, 2) for every directed edge $(x, y) \in E$, $(x, y) \in V_1 \times V_2$. If $E = V_1 \times V_2$, an episode G is called a *proper bipartite episode*. Obviously, vertices in a bipartite episode G are separated into V_1 and V_2, and we could regard them as generalizations of the source vertex and the sink vertex of diamond episodes. This indicates that the same way is applicable for bipartite episodes by defining $\sqcap$ between sets of tress. Fortunately, [9] gives the definition $\sqcap$ for sets of graphs.

$$\{t_1, \ldots, t_k\} \sqcap \{s_1, \ldots, s_m\} \equiv \mathrm{MAX}_{\sqsubseteq_T} \left(\bigcup_{i,j} (\{t_i\} \sqcap \{s_j\}) \right),$$

where $\mathrm{MAX}_{\sqsubseteq_T}(S)$ returns only maximal elements in S with respect to $\sqsubseteq_T$. Since our generalized subtree isomorphism is basically a special case of that for graphs, we can also apply this meet operation. This example suggest that if we have some background knowledge concerning a partition of V, it can be taken into account for δ and $(D, \sqcap)$ in a similar manner of diamond and bipartite episodes.

7 Conclusions and Future Work

In this paper we propose a pattern structure for diamond episodes based on an idea used in graph kernels and projections of pattern structures. Since we do not directly compute graph matching operations we conjecture that our computation could be efficient. With a slight modification of $\sqcap$, our method is also applicable for many classes of episodes, not only for diamond patterns as we mentioned above. Based on our pattern structure, we discussed summarization by using mined pattern concepts and show small examples and experimental results.

Since problems of this type are unsupervised and there is no common way of obtaining good results and of evaluating whether or not the results are good. It would be interesting to study more about this summarization problem based on concept lattices by taking into account theoretical backgrounds such as probabilistic distributions. In our future work, we try to analyze theoretical aspects on summarization via pattern structures including the wildcard $\star$ and its optimization problem to obtain compact and interesting summarization of many patterns based on our important merit of a partial order $\sqsubseteq$ between descriptions.

Acknowledgments

This work was supported by Grant-in-Aid for JSPS Fellows (26·4555) and JSPS KAKENHI Grant Number 26280085.

References

1. Al Hasan, M., Chaoji, V., Salem, S., Besson, J., Zaki, M.: Origami: Mining representative orthogonal graph patterns. In: Proc. of the 7th ICDM. pp. 153–162 (2007)
2. 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)
3. De Bie, T.: Maximum entropy models and subjective interestingness: an application to tiles in binary databases. Data Mining and Knowledge Discovery 23(3), 407–446 (2011)
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., Mielik ainen, T.: Tiling databases. In: Proc. of the 7th DS. pp. 278–289 (2004)
7. Katoh, T., Arimura, H., Hirata, K.: A polynomial-delay polynomial-space algorithm for extracting frequent diamond episodes from event sequences. In: Proc. of the 13th PAKDD. pp. 172–183. Springer Berlin Heidelberg (2009)
8. Kaytoue, M., Kuznetsov, S.O., Napoli, A.: Revisiting Numerical Pattern Mining with Formal Concept Analysis. In: Proc. of the 24th IJCAI (2011)
9. Kuznetsov, S.O., Samokhin, M.V.: Learning closed sets of labeled graphs for chemical applications. In: Proc. of the 15th ILP, pp. 190–208 (2005)
10. Lloyd, J.W.: Foundations of Logic Programming. Springer-Verlag New York, Inc.
11. Mannila, H., Toivonen, H., Inkeri Verkamo, A.: Discovery of frequent episodes in event sequences. Data Mining and Knowledge Discovery 1(3), 259–289 (1997)
12. Merwe, D., Obiedkov, S., Kourie, D.: AddIntent: A New Incremental Algorithm for Constructing Concept Lattices. In: Proc. of the 2nd ICFCA. pp. 372–385 (2004)
13. 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)
14. Shervashidze, N., Schweitzer, P., van Leeuwen, E.J., Mehlhorn, K., Borgwardt, K.M.: Weisfeiler-lehman graph kernels. Journal of Machine Learning Research 12, 2539–2561 (2011)
15. Stumme, G., Taouil, R., Bastide, Y., Pasquier, N., Lakhal, L.: Computing iceberg concept lattices with titanic. Data & Knowledge Engineering 42(2), 189–222 (2002)
16. Uno, T.: An efficient algorithm for solving pseudo clique enumeration problem. Algorithmica 56(1), 3–16 (Jan 2010)
17. Vreeken, J., van Leeuwen, M., Siebes, A.: Krimp: mining itemsets that compress. Data Mining and Knowledge Discovery 23(1), 169–214 (2011)
18. Xin, D., Cheng, H., Yan, X., Han, J.: Extracting redundancy-aware top-k patterns. In: Proc. of the 12th KDD. pp. 444–453. ACM (2006)