Summarizing Sequential Data with Closed Partial Orders *

Gemma C. Garriga†

Abstract
In this paper we address the task of summarizing a set of input sequences by means of local ordering relationships on items occurring in the sequences. Our goal is not mining these structures directly from the data, but going beyond the idea of closed sequential patterns and generalize it into a novel notion of closed partial order. We will show that just a simple (but not trivial) post-processing of the closed sequences found in the data leads to a compact set of informative closed partial orders. We analyze our proposal not only algorithmically but also theoretically, by showing the connection with Galois lattices. Finally, we illustrate the approach by applying it to real data.

General Terms. Closed partial orders, sequence analysis, post-processing closed sequential patterns.

1 Introduction

Mining sequences of events is an important data mining task with broad applications in business, web mining, computer intrusion detection, DNA sequence analysis and so on. The problem was first introduced in [1] as a problem of mining frequent sequential patterns in a set of sequences, and since then, it has been extensively studied (e.g., algorithms like SPADE [19] or PrefixSpan [13] among others). Unfortunately, one problem of this sequential pattern mining task arises when considering a very low support in the algorithms or when mining very long sequences; in these cases, the number of frequent patterns is usually too large for a thorough examination and the algorithms face several computational problems. A proper solution to this problem is recently proposed in some papers, such as [15, 16, 17], and it consists on mining just a compact and more significative set of patterns called the *closed sequential patterns* (or closed sequences). These closed sequential patterns are defined to be "stable" in terms of support, that is, they are maximal sequences among those others having the same support in the database.

The idea of mining just closed sequential patterns instead of all frequent patterns stems from the parallel case of mining closed itemsets in a binary database ([12, 18]). The foundations of closed itemsets are based on the mathematical model of concept lattices ([5, 6]): a closure operator is defined by using the properties of the Galois connection, and from there, one can draw a lattice of formal concepts. Then, it can be proven that the set of closed itemsets is necessary and sufficient to capture all the information about frequent itemsets and association rules in the unordered context. Moving to the sequential case again, a recent work in [7] proves that the set of closed sequential patterns mined by existing algorithms [15, 16, 17] can be formalized in terms of a closure operator as well.

In general, dealing with closed patterns is currently an interesting topic in data mining since it provides a more compact set of patterns. However, we consider that there are still some criticisms to be done about the closed sequences: mainly, the number of those patterns can be still quite large due to the combinatorial nature of the problem, and it is not clear how they can be useful to the final user once we have mined them.

1.1 Goals of this Work In this paper we propose a way to handle these resulting closed sequences so that they provide useful information of our data. We are not focusing here on algorithmic solutions for finding closed sequential patterns, and we rely on current proposals such as TSP [15], BIDE [16] or CloSpan [17]; our intention is not contributing to the efficiency of existing algorithms, but to the post-processing of closed sequences once we have mined them. Our goal is to outcome with a new notion of partial orders that can be obtained out of the closed sequences, in such a way that (1) it advances in the summarization of sequential data; (2) it has a sound theory supporting it; and (3) it can be implemented with efficient algorithms without accessing the input data, just the set of closed sequences. Finally, we will show that these partial orders represent indeed the closure of hybrid episodes introduced in [11], and they can be seen also as complementary to other works of mining episodes.

1.2 Paper Overview The rest of the paper is organized as follows. In section 2 we present some basic definitions of the frequent closed sequence mining. Section 3 motivates our intention of going beyond closed sequences and defines our post-processing approach for generating partial orders out of the set of closed se-

*Supported by MCYT TIC 2002-04019-C03-01 (MOISES)
†Universitat Politècnica de Catalunya, Barcelona, Spain

quences. Sections 4 and 5 develop algorithmically and theoretically the two steps of the proposal; section 6 discusses other algorithmic issues to complete the current proposal. Finally, section 7 evaluates this work with experiments, and section 8 discusses the relation of our final partial orders with the hybrid episodes of [11]. In section 9 we conclude the study.

2 Preliminaries

Let $\mathcal{I} = \{i_1, \ldots, i_n\}$ be a finite set of items. In the classical formalization given by [1], *sequences* are ordered lists of itemsets. The input data we are considering consists of a database of ordered transactions $\mathcal{D} = \{t_1, t_2, \ldots t_n\}$ that we model as a set of sequences, i.e. each transaction t_i is in fact a sequence, also called *input sequence*. Our notation for the component itemsets of a given sequence will be $s = \langle (I_1)(I_2)\ldots(I_n)\rangle$, where each $I_i \subseteq \mathcal{I}$ and I_i occurs before itemset I_j if $i < j$. Note that each I_i may contain several items that occur simultaneously; e.g. $\langle (AC)(B)\rangle$, meaning that items A and C come at the same time but always before item B. The universe of all sequences is noted with $\mathcal{S}$. An example of such data $\mathcal{D}$ is presented in figure 1, where each itemset I_i contains only one single item. We choose this simplification to make this first example more clear, and because we consider that these single-item sequences model popular types of data such as DNA, Web click streams, command histories of Unix users and so on. However, our proposals will work also when having sequences of subsets of items, and in section 6 of this paper we will follow up an example with simultaneity.

Seq id	Input sequences
t_1	$\langle (C)(B)(C)(A)(C)\rangle$
t_2	$\langle (C)(B)(A)(C)(C)(C)(A)\rangle$
t_3	$\langle (A)(C)(A)(C)(C)(A)(A)(A)\rangle$
t_4	$\langle (C)(A)(C)\rangle$

Figure 1: Collection of data $\mathcal{D}$

Some basic operations on sequences can be defined in a general way as follows. Sequence $s = \langle (I_1)\ldots(I_n)\rangle$ is a *subsequence* of another sequence $s' = \langle (I'_1)\ldots(I'_m)\rangle$ if there exist integers $j_1 < j_2 \cdots < j_n$ such that $I_1 \subseteq I'_{j_1}, \ldots, I_n \subseteq I'_{j_n}$. We note it by $s \subseteq s'$. For example, $s = \langle (C)(B)(A)\rangle$ is a subsequence of the first and second input sequences from figure 1, $s \subseteq t_1$ and $s \subseteq t_2$. For later formalization purposes it will be necessary to keep track of the identifiers of those input sequences where a certain s is contained, named its tid list, $tid(s)$; e.g., $tid(\langle (C)(B)(A)\rangle) = \{t_1, t_2\} = \{1, 2\}$ in figure 1. Then, the *support* of a sequence s is the number of input sequences where s is contained, $support(s) = |tid(s)|$; e.g., the support of $\langle (C)(B)(A)\rangle$ is 2.

The *intersection* of a set of sequences $s_1, \ldots, s_n \in \mathcal{S}$ is the set of *maximal* subsequences contained in all the s_i. Note that the intersection of a set of sequences, or even the intersection of two sequences, is not necessarily a single sequence. For example, the intersection of the two sequences $s = \langle (A)(C)(B)\rangle$ and $s' = \langle (A)(B)(C)\rangle$ is the set of sequences $\{\langle (A)(C)\rangle, \langle (A)(B)\rangle\}$: both are contained in s and s', and among those having this property they are maximal; all other common subsequences are not maximal since they can be extended to one of these. The maximality condition discards redundant information since the presence of, e.g., $\langle (A)(B)\rangle$ in the intersection already informs of the presence of each of the itemsets (A) and (B).

The *head* of a sequence $s = \langle (I_1)\ldots(I_n)\rangle$ up to a position j s.t. $1 \leq j \leq n$, is noted by $head(s, j) = \langle (I_1)\ldots(I_j)\rangle$. Similarly, the *tail* of a sequence from position j s.t. $1 \leq j \leq n$, is noted by $tail(s, j) = \langle (I_j)\ldots(I_n)\rangle$. The *concatenation* of two sequences will be noted by $s \diamond s'$.

2.1 Mining Closed Sequential Patterns

Tipically, associated to this discrete sequential data there is the problem of mining frequent sequences, that is, those subsequences in $\mathcal{D}$ whose support is over a user-specified threshold. Unfortunately, the performance of the algorithm degrades when using a very low support or having a dense database. Some recent works, such as [15, 16, 17], propose to mine frequent closed sequences instead.

A sequence s is *closed* in input data $\mathcal{D}$ if s is maximal in the set of transactions where it is contained, that is, it cannot be extended. More formally:

Definition 2.1. (Closed Sequence) *A sequence $s \in \mathcal{S}$ is closed for data $\mathcal{D}$ if there exists no sequence s' with $s \subset s'$ s.t. $support(s) = support(s')$.*

For instance, taking data from figure 1, sequence $\langle (A)(C)\rangle$ is not closed since it can be extended to $\langle (C)(A)(C)\rangle$ in all the input sequences where it belongs. However, $\langle (C)(B)(A)(C)\rangle$ or $\langle (C)(C)(C)\rangle$ are closed sequences in $\mathcal{D}$. In other words, closed sequences are "stable" in terms of support since they are maximal among those having the same support. The set of all closed sequential patterns of data from figure 1 is shown in figure 2. Usually, a minimum support condition is provided by the user to mine only those closed sequences up to the threshold; here, given that data in figure 1 is small enough, we will suppose that the threshold is set to zero for this ongoing example.

Tid list	Closed Sequential Patterns
$\{1,2,3,4\}$	$\langle(C)(A)(C)\rangle$
$\{1,2,3\}$	$\langle(C)(C)(A)\rangle$
$\{1,2,3\}$	$\langle(C)(C)(C)\rangle$
$\{2,3\}$	$\langle(A)(C)(C)(C)(A)\rangle$
$\{2,3\}$	$\langle(C)(A)(C)(C)(A)\rangle$
$\{1,2\}$	$\langle(C)(B)(A)(C)\rangle$
$\{1,2\}$	$\langle(C)(B)(C)(A)\rangle$
$\{1,2\}$	$\langle(C)(B)(C)(C)\rangle$
$\{1\}$	$\langle(C)(B)(C)(A)(C)\rangle$
$\{2\}$	$\langle(C)(B)(A)(C)(C)(C)(A)\rangle$
$\{3\}$	$\langle(A)(C)(A)(C)(C)(A)(A)(A)\rangle$

Figure 2: Set of all closed sequences and their tid lists

3 Discussion and Motivation

Frequent closed sequential patterns represent the most informative total orders in $\mathcal{D}$ with respect to support. So, apart from reducing the algorithmic overhead, this final set of closed patterns is useful in many ways: (1) the user needs to examine fewer patterns obtained as an output of the mining algorithms; (2) hitting with the right minimum support threshold is not so important; for example, mining all the subsequences in $\mathcal{D}$ with a threshold close to zero is unrealistic and it does not provide useful information about the data, but the set of all closed sequences is not so dramatic and still gives an overall idea of the whole database; and (3) closed patterns have a sound theoretical background based on formal concept analysis (see [7]), which provides several important properties and formalizations.

Therefore, the set of closed sequences provides a more compact set of patterns that keeps the same information as frequent sequences. However, many questions arise: what do these sequences say about our data $\mathcal{D}$? what to do with these closed sequential patterns once we have mined them? how to go beyond closed sequential patterns?

We consider that the set of closed sequential patterns does not represent all the particularities hidden in the sequential data. Formally, it can exist two closed sequences s and s' such that they occur in the same transactions, so that $support(s) = support(s')$, but $s \not\subseteq s'$ and $s' \not\subseteq s$. In other words, contrary to the case of closed itemsets in binary data ([12, 18]), here there is no unique representative closed pattern for a given set of ordered transactions. By way of example, the closed sequences $\langle(A)(C)(C)(C)(A)\rangle$ and $\langle(C)(A)(C)(C)(A)\rangle$ from figure 2 occur exactly in the same transactions but none of them can be considered "better" than the other, they simply *coexist* together. This fact cannot be captured by the unidimensional representation of the set of closed sequences.

We want to go beyond the notion of closed sequential patterns. Our goal is to generalize this idea of compacting the information as much as possible so that we can produce not just fewer patterns, but also more informative ones. Next, we will show how these closed sequences coexisting together lead to a novel notion of closed partial order that summarize the data in a compact way. We formalize our proposal:

1. First, grouping closed sequential patterns occurring together in a maximal set of transactions. We will see that this task has not a direct solution since some closed sequences can coexist in several groups and also, we want to make these groups nonredundant and maximal.

2. Second, constructing a new notion of closed partial orders out of those groups, without the need of accessing again the data $\mathcal{D}$.

We will see that this process is supported by the mathematical theory of formal concept analysis since the set of maximal sequences in a maximal set of transactions is a closure system. This ensures the good properties of the obtained results. In next two sections, we describe with detail the goals of our approach and we provide efficient algorithms to implement each one of the steps.

4 Grouping Closed Sequential Patterns

The first step of our proposal is to make nonredundant groups of sequences coexisting together in the same maximal set of transactions. Formally, we state this problem as: given the set of frequent closed sequential patterns $CS = \{s_1, s_2, \ldots, s_n\}$ mined by any of the existing algorithms, we want to output a list of *valid* pairs (S, T).

DEFINITION 4.1. *A valid pair (S, T) is one where: $S \subseteq CS$ is a* nonredundant *set of closed sequences, whose tid lists are at least T; and $T \subseteq \mathcal{D}$ is the* maximal *set of transactions where all $s \in S$ are contained.*

We say that a set S of closed sequences is *nonredundant* when for all $s, s' \in S$ s.t. $s \neq s'$ we have that $s \not\subseteq s'$ and $s' \not\subseteq s$. By computing the list of valid pairs (S, T) from CS, we get nonredundant groups S of closed sequences where all $s \in S$ is contained in all $t \in T$, and there is no other set of transactions T' with $T \subset T'$ where sequences in S still coexist together. An obvious observation then is that the set of transactions T of a valid pair (S, T) will correspond to the maximal tid list of one of the sequences $s \in S$, that is, $T = \max\{tid(s) | \forall s \in S\}$. Otherwise T would not be maximal as we want, since all the sequences in S must have tid lists at least T by definition.

E.g., sequences $\langle(A)(C)(C)(C)(A)\rangle$ and $\langle(C)(A)(C)(C)(A)\rangle$ from figure 2, will be fitted into the same set S since they appear exactly in the same input sequences $T = \{2, 3\}$. Any other closed sequence whose tid list is at least T (i.e. coexisting still in transactions T) would not fit in S, since it would turn it redundant. Moreover, the set $T = \{2, 3\}$ is a maximal set of transactions for this S as well, that is, for any other larger set of transactions we have that the sequences in S coexist together. A complete example of the desired groups for the closed sequences in 2, is shown in figure 3. Note that there is no valid pair involving the set of transactions $T = \{1, 3\}$: this T is not maximal for the set of closed sequences whose tid list is at least $T = \{1, 3\}$ (and no closed sequential pattern has a tid list coinciding with this T).

T	S
$\{1, 2, 3, 4\}$	$\{\langle(C)(A)(C)\rangle\}$
$\{1, 2, 3\}$	$\{\langle(C)(A)(C)\rangle, \langle(C)(C)(A)\rangle, \langle(C)(C)(C)\rangle\}$
$\{2, 3\}$	$\{\langle(A)(C)(C)(C)(A)\rangle, \langle(C)(A)(C)(C)(A)\rangle\}$
$\{1, 2\}$	$\{\langle(C)(B)(A)(C)\rangle, \langle(C)(B)(C)(A)\rangle, \langle(C)(B)(C)(C)\rangle\}$
$\{1\}$	$\{\langle(C)(B)(C)(A)(C)\rangle\}$
$\{2\}$	$\{\langle(C)(B)(A)(C)(C)(C)(A)\rangle\}$
$\{3\}$	$\{\langle(A)(C)(A)(C)(C)(A)(A)(A)\rangle\}$

Figure 3: Groups of closed sequences occurring together

Obviously, an initial algorithmic naive approach is to group all those closed sequences in CS with the same tid list, that is, create each pair (S, T) so that all $s \in S$ has the same tid list and $T = tid(s)$. In this way we ensure that the set T will be maximal. However, the set S created in the naive way may be incomplete, with some missing element. The following property formalizes this idea.

PROPOSITION 4.1. *Let (S, T) be a valid pair, then we have that* $S = \bigcap_{t \in T} t$.

Proof. If T is a set of transactions from a given valid pair, then the intersection of transactions $t \in T$ returns a set of sequences S s.t. for all $s \in S$ we have that $tid(s)$ is at least T. Moreover, all sequences $s \in S$ will be closed and nonredundant by definition of the intersection operation, that always keeps only maximal subsequences.

Proposition 4.1 simply states that the nonredundat group of closed sequences coexisting in a set of transactions T must necessarily coincide with the intersection of those transactions. Note that the reverse implication of the proposition does not hold, because we cannot ensure the maximality of the set of transactions T. Then, a naive gropuping such as $S_1 = \{\langle(A)(C)(C)(C)(A)\rangle, \langle(C)(A)(C)(C)(A)\rangle\}$ and $T = \{2, 3\}$ of our example, is valid because it coincides with the intersection of second and third input sequences of $\mathcal{D}$. However, another naive group such as $\{\langle(C)(C)(A)\rangle, \langle(C)(C)(C)\rangle\}$ and $T = \{1, 2, 3\}$, does not form a valid pair because the intersection of first, second and third input sequences contains one more closed sequence; the proper family of sequences of this valid pair should be $S_2 = \{\langle(C)(A)(C)\rangle, \langle(C)(C)(A)\rangle, \langle(C)(C)(C)\rangle\}$. Indeed, the closed sequence $\langle(C)(A)(C)\rangle$ is contained in all the transactions of the database; then, it always occurs simultaneously with any other sequence, and it should be included in any S as long as it does not make it redundant. Because of this, $\langle(C)(A)(C)\rangle$ belongs to S_2 as an element, but not to S_1, where it is already included as a subsequence of $\langle(C)(A)(C)(C)(A)\rangle$.

Of course, we want to output the valid list of pairs (S, T) without directly intersecting the transactions of the database, just by grouping the sequences in CS. But there can be closed sequences that belong to several sets S as long as S is not redundant. The general property that follows from this reasoning is expressed by the following lemma.

LEMMA 4.1. *Given two valid pairs (S', T') and (S, T), if $T \subseteq T'$ then for all $s' \in S'$ there exists $s \in S$ s.t. $s' \subseteq s$.*

Proof. For all $s' \in S'$ we have that s' is subsequence of t', for all $t' \in T'$ (proposition 4.1); in particular, if $T \subseteq T'$, then we have that s' is subsequence of t for all $t \in T$, and thus there exists $s \in S$ s.t. $s' \subseteq s$.

4.1 Algorithmic Analysis Conceptually, lemma 4.1 provides a way to organize the list of valid pairs into a graph where each node is a pair (S, T), and there is an edge between each node and its predecessors. The predecessors of a node (S, T) are those pairs (S', T') s.t. $T \subseteq T'$. The property stated in lemma 4.1 must hold between each node and their predecessors.

In figure 4 we graphically depict the mentioned conceptual graph for the valid pairs of figure 3. Each node represents a valid pair (S, T), and here we depict only those edges connecting immediate predecessors: (S', T') is an immediate predecessor of (S, T) if $T \subseteq T'$ and there is no other (S'', T'') s.t. $T \subset T''$ and $T'' \subset T'$. E.g. the node $\{\langle(C)(B)(C)(A)(C)\rangle\}$ has one immediate predecessor, which is $\{\langle(C)(B)(A)(C)\rangle, \langle(C)(B)(C)(A)\rangle, \langle(C)(B)(C)(C)\rangle\}$. We will note immediate predecessors by relation $\preceq$: $(S', T') \preceq (S, T)$. Notice that if lemma 4.1 holds for immediate predecessors of a given node, then it will hold for the rest of predecessors as well (see figure 4).

This conceptual graph will be used as a tool to

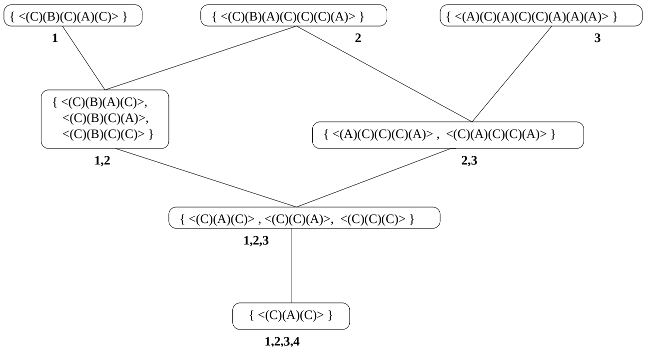

Figure 4: Conceptual graph of pairs (S, T)

analyze our algorithmic proposal: it simplifies the pseudocode and it eases the complexity analysis. However, the final implementation does not necessarily need to recreate such structure in memory, and a wise use of lists and indexes turns out to be enough to come up with the right groupings. The pseudocode to obtain valid pairs is presented in algorithm 1.

Algorithm 1 Grouping Closed Sequential Patterns

Input: List CS of closed sequential Patterns
Output: List of pairs (S, T)

1: Sort CS in descending order by tid list;
2: **while** $CS \neq \emptyset$ **do**
3: $S \leftarrow$ Next sequences $s \in CS$ with same tid list;
4: $T \leftarrow tid(s)$, for some $s \in S$;
5: **for each** $(S', T') \preceq (S, T)$ **do**
6: **for each** $s' \in S'$ **do**
7: **if** $s' \nsubseteq s,\ \forall s \in S$ **then** $S \leftarrow S \cup \{s'\}$ **end if**
8: output (S, T);
9: **end for**
10: **end for**
11: **end while**

The idea of algorithm 1 is simple: lines 3-4 perform naively by getting groups of closed sequences in CS with same tid list. Then, lines 5-11 complete the set S with sequences already belonging to immediate predecessors that should be also in S. Considering that CS is ordered in descending order by the tid list of its elements, then the algorithm traverses the conceptual graph bottom-up in a breadth-first fashion.

Notice again the fact that it is only necessary to look for immediate predecessors of a set S to make lemma 4.1 hold. The algorithmic proposal is bounded by $O(n \cdot m \cdot k^2)$, where we consider n to be the number of closed sequences in CS, m is the maximum number of immediate predecessors for a node (S, T), and k is the maximum number of closed sequences belonging to immediate predecessors of S.

4.2 Theoretical Analysis An important theoretical consideration to be done at this point is that the set of closed sequential patterns in S of valid pairs (S, T) can be formalized in terms of a closure operator named Δ, presented in [7]. Basically, a closure operator of any fixed universe is one that satisfies the three basic closure axioms: monotonicity, extensivity and idempotency. The formal operator Δ developed in [7] works with *sets of sequences*.

Broadly speaking, this resulting Δ stems from the composition of two derivation operators forming a Galois connection and it works as follows: given $\mathcal{D}$, the closure $\Delta(S)$ of a set of sequences S, includes all the maximal sequences that are present in all transactions having all sequences in S; for example, in data from figure 1 we have $\Delta(\{\langle(A)(C)(C)\rangle\}) = \{\langle(A)(C)(C)(C)(A)\rangle, \langle(C)(A)(C)(C)(A)\rangle\}$ because all the maximal subsequences contained in those transactions where $\{\langle(A)(C)(C)\rangle\}$ belongs are $\langle(A)(C)(C)(C)(A)\rangle$ and $\langle(C)(A)(C)(C)(A)\rangle$. Then, we say that closed sets of sequences are those coinciding with their closure, that is, $\Delta(S) = S$.

Given proposition 4.1 and lemma 4.1, it is possible to prove that the set of closed sequences S from each pair valid (S, T) indeed fulfills $\Delta(S) = S$. So, sets S from valid pairs are also a closed set of sequences according to Δ. This characterization carries several consequences. The first important consequence is that the conceptual graph described in subsection 4.1 can be considered a Galois lattice ([5, 6]) under these certain conditions: (1) when nodes of the graph are closed under intersection; and (2) there exists a supremum and infimum for any pair of elements. Broadly, first condition means that when intersecting two sets of sequences S_1 and S_2 of to two different nodes, we get another set of sequences, $S_3 = S_1 \bigcap S_2$, belonging to another node of the same graph. That is, the intersection of two sets of sequences from valid pairs returns another set of sequences from another valid pair. Here, the intersection of two sets of sequences $S_1 \bigcap S_2$ is defined as the cross intersection of all $s_1 \in S_1$ with all $s_2 \in S_2$. Similarly, the set of transactions T of valid pairs must be also close under intersection to be considered a Galois lattice. For example, the graph depicted in figure 4 is closed under intersection, so, if we added an artificial top connected to the upper vertices of the graph, then we would get a Galois lattice. The conceptual graph of valid pairs may not be always closed under intersection, then, the missing nodes must be added to form a closure system. More details can be followed from [7].

The second consequence of having this characterization in terms of Δ, is that it is possible to derive a notion of deterministic association rules from valid pairs (S, T) and the generators of S. We say that a set S' is a generator of S when $\Delta(S') = S$ and $S \neq S'$. Then, the theory of associations for this ordered context can be completely formulated by considering implications of the form $S' \rightarrow S$. Actually, a recent work in [3] shows that these implications axiomatize the Horn theory for ordered data.

5 Obtaining Closed Partial Orders

The second step of our proposal is to obtain a compact representation from each valid pair (S, T). Our starting point is to realize that each $s \in S$ is in fact a total order compatible with all transactions $t \in T$. Since sequences in S coexist together in T, it should be possible to derive a partial order describing the set of transactions T by properly combining all $s \in S$. To get an initial intuition of this idea we show in figure 5 the desired transformation that will turn the sets S from figure 3 into partial orders; e.g., the set $S = \{\langle(A)(C)(C)(C)(A)\rangle, \langle(C)(A)(C)(C)(A)\rangle\}$ is converted into a partial order compatible with transactions $T = \{2, 3\}$. Basically, we are creating a partial order out of S, without adding new restrictions among the different items and still respecting all the restrictions given by each $s \in S$. We may think that this is an easy task, but the way of overlaping the positions of a set of sequences S to get a partial order still compatible with a set of transactions T is not direct, specially when having repeated items.

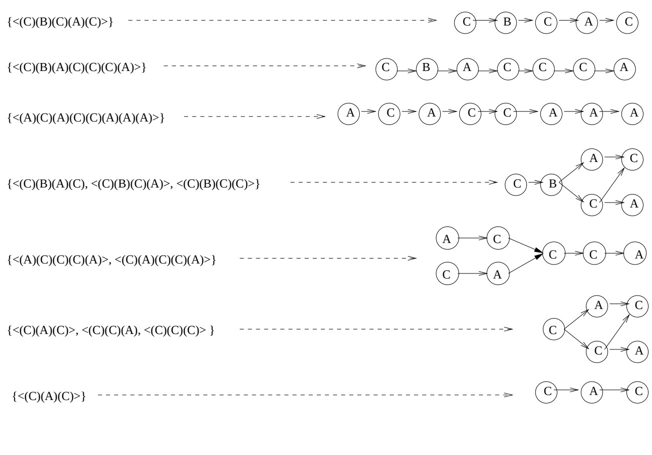

Figure 5: Partial orders from (S, T)

To make this goal more formal we introduce some basic definitions. A *partial order* can be modelled as a triple $p = (V, E, l)$ where V is the set of vertices; $E \subseteq V \times V$ is the set of edges such that the relation on V established by edges in E is reflexive, antisymmetric and transitive; and l is the labelling function mapping each vertex into a set of items, i.e. $l : V \rightarrow 2^{\mathcal{I}}$. In our proposed example of figure 5, this labelling function simply maps each vertice to a single item, but indeed a node can be labelled with a set of items when considering sequences of itemsets. The *transitive reduction* of $p = (V, E, l)$ is the smallest relation resulting from deleting those edges in E that come from transitivity. Partial orders will be graphically depicted here by means of its transitive reduction to make them more understandable, but of course, all edges of the transitive closure are present in E. The graphical representation of partial orders is particularly useful for displaying results: we display a poset by using arrows between the connected labelled vertices, and the symbol $\|$ (parallel) to indicate trivial order among the different components of a partial order.

We say that a partial order $p = (V, E, l)$ is *compatible* with an (input) sequence s if: $\forall u \in V$ we have that $l(u)$ is in s; and, $\forall (u, v) \in E$ we have that $\langle(l(u))(l(v))\rangle \subseteq s$. The support of a partial order is the number of input transactions that is compatible with.

Given (S, T) our goal is to generate a partial order $p = (V, E, l)$ s.t. for all $s' \subset s \in S$, we have that s' is included in p; and p is still compatible with all transactions in T. The first condition forces the partial order p to respect all the restrictions given by each $s \in S$, and the second condition ensures that no extra edges will be added between vertices (next lemma 5.1 will formalize this idea). This partial order will summarize the set of transactions T in the most specific way. In subsection 5.1 we will provide more clear formalizations about this notion. We want the partial order generated from a pair (S, T) to be exactly compatible with transactions in T.

One way to get such structure out of (S, T) is by considering a partial order whose *maximal paths are exactly* defined by sequences $s \in S$. We define a path from a partial order $p = (V, E, l)$ as a sequence $\langle(I_1) \ldots (I_n)\rangle$ such that there is an equivalent list of different nodes $u_1, \ldots, u_n$ in V that $(u_j, u_{j+1}) \in E$ and $l(u_j) = I_j$ and $l(u_{j+1}) = I_{j+1}$. E.g. in figure 5, groups of sequences S define the maximal paths of the new partial orders, and the maximal paths of these partial orders coincide exactly with the sequences in the set S. Moreover, if we considered different paths not included in S, then p would not be compatible with all transactions in T, as it shows the following lemma.

Lemma 5.1. *Given a valid pair (S, T) and a partial order p, we have that:*

(1) *if p has maximal paths exactly defined by sequences in S, then p is only compatible with transactions in T.*

(2) *if p has any paths not included in S, then p is not compatible with transactions T.*

Proof. Proving part (1) is easy: if p has maximal paths defined by sequences in S of a valid pair (S, T), then p will be compatible with transactions in T, since all $s \in S$ is a subsequence of all $t \in T$ (prop. 4.1). Moreover, p cannot be compatible with more transactions not considered in T, because by definition of valid pair we have that T is a maximal set of transactions for S. So, it does not exist an larger set of transactions T' where sequences in S are all of them included at a time.

Part (2) of the lemma can be proved as follows. If there exists a sequence s' representing path of p s.t. $s' \notin S$, then we can rewrite it as $s' \notin \bigcap t, \forall t \in T$ (prop. 4.1). This implies that s' is not a subsequence of some $t \in T$, and so, the considered p cannot be compatible with all transactions T, as stated by the lemma.

According to lemma 5.1, given (S, T) it is not possible to create a partial order which is still compatible with T and includes more paths not considered in S. Following this idea, we will generate partial orders whose maximal paths are exactly defined by sequences in S. Then, we have that (1) p will be compatible with all transactions $t \in T$, since each $s \in S$ appears in all $t \in T$; (2) p is not having new edges representing restrictions not considered in S; and (3) p obviously respects each $s \in S$, since for all $s' \subset s$ we have that s' will be included in p. In subsection 5.2 we will detail theoretical results making this partial order closed for data $\mathcal{D}$.

5.1 Algorithmic Analysis The point is now how to match positions of sequences in S so that they form the maximal paths of a partial order. This is not a direct task since sequences can be overlapped in several ways so that the resulting partial order still has maximal paths in S. For example, from the set of closed sequences $S = \{\langle(A)(C)(C)(C)(A)\rangle, \langle(C)(A)(C)(C)(A)\rangle\}$ there are several partial orders whose maximal paths are exactly sequences in S: starting from the fully independent parallelization of both sequences, or just matching the last item (A) of both sequences, or just matching the two last items $(C)(A)$ of both sequences, and so on; but we are interested only in the partial order generated as in figure 6, which is matching the last three positions $(C)(C)(A)$ from both sequences.

In other words, from all the partial orders whose maximal paths are exactly sequences in S, we are interested in the most specific one. We define that

Figure 6: Transformation into a partial order

a partial order $p' = (V', E', l')$ is more *specific* than another partial order $p = (V, E, l)$, noted by $p \trianglelefteq p'$, if there exists an injective mapping $h : V \to V'$ such that preserves labels (that is, $l \subseteq l' \circ h$), and $(u, v) \in E \Rightarrow (h(u), h(v)) \in E'$.

To construct the most specific partial order p we will match as many positions as possible from sequences $s \in S$. Yet there is still the problem of identifying which positions must be matched. Note that not all positions having same label are good to be overlapped: e.g. taking again the set $S = \{\langle(A)(C)(C)(C)(A)\rangle, \langle(C)(A)(C)(C)(A)\rangle\}$, the first (C) of $\langle(A)(C)(C)(C)(A)\rangle$ will not be overlapped with any (C) of the other sequence, otherwise we would get a partial order having more paths than the ones included in S. So, when matching positions of sequences in S, the algorithm has to take care of not adding new paths to p that are not in S (and so, keep the properties of lemma 5.1). The idea is that, given two sequences $s, s' \in S$, two positions can be matched as long as they are path preserving with respect to all sequences in S.

Definition 5.1. (Path Preserving Positions) *Given a set of sequences S and let $s, s' \in S$ be two sequences s.t. $s = \langle(I_1)\dots(I_i)\dots(I_n)\rangle$ and $s = \langle(I'_1)\dots(I'_j)\dots(I'_m)\rangle$, then positions i of s and j of s' are path preserving if,*

- *$I_i = I'_j$; and,*
- *$head(s, i) \diamond tail(s', j+1) \subseteq s''$, for some $s'' \in S$; and,*
- *$head(s', j) \diamond tail(s, i+1) \subseteq s''$, for some $s'' \in S$.*

Then, we say that position i of s matches with position j of s'; we note it by $p[i] \sim q[j]$.

This definition ensures that when matching different positions of sequences in S, any new possible paths that can be created by this overlapping will be always included in S. Note that this operation is symmetric, i.e. if $p[i] \sim q[j]$ then $q[j] \sim p[i]$; and also transitivite as it is showed in the following proposition.

Proposition 5.1. (Transitivity) *Given a valid pair (S, T) and let $s, s', s'' \in S$, if $s[i] \sim s'[j]$ and $s'[j] \sim s''[k]$, then $s[i] \sim s''[k]$.*

Proof. For a given valid pair (S, T) we have that $S = \bigcap t, \forall t \in T$ (prop. 4.1), which makes this set S consistent for transitivity property. In particular, if $s[i] \sim$

$s'[j]$ and $s'[j] \sim s''[k]$, then we have that S contains at least these following three sequences among others: $head(s, i) \diamond tail(s, i + 1)$ (i.e. sequence s), and $head(s', j) \diamond tail(s, i+1)$, and $head(s', j) \diamond tail(s'', k+1)$. The presence of these three sequences in S forces the sequence $head(s, i) \diamond tail(s'', k + 1)$ to be also present in the intersection of all $t \in T$ as well. Similarly, we can justify that S also contains the other necessary path $head(s'', k) \diamond tail(s, i+1)$; thus, reaching the conclusion that $s[i] \sim s''[k]$.

Transitivity also ensures that a position from a sequence $s \in S$ cannot be matched with two different positions from $s' \in S$; otherwise, we would get a cycle and sequences in S would not be valid. The algorithmic solution for this problem finds the matching positions of a group of sequences S, of a valid pair (S, T). The idea is checking definition 5.1 for all positions of the sequences in S. It is possible to improve the algorithm by using the transitivity property, and we rely on the fact that the number of sequences in S and the length of these sequences is quite reasonable.

Algorithm 2 Matching Positions of a Set of Sequences

Input: A group of sequences S
Output: A list of positions to be joined

```
for all s, s' ∈ S s.t. s ≠ s' do
  for all positions i of s and j of s' do
    output whether p[i] ∼ q[j];
  end for
end for
```

Algorithm 2 is the pseudocode that decides which positions must be matched of a group of closed sequences S. To get the most specific partial order, all the path preserving positions must be overlapped. Of course, this algorithm must be used over each set of closed sequences S obtained after algorithm 1, leading to different partial orders for each (S, T). For example, in figure 5, each poset is the result of matching all the path preserving positions of sequences in S.

5.2 Theoretical Analysis

In this section we want to show that the partial orders obtained by the proposed approach are indeed closed partial orders in $\mathcal{D}$.

Definition 5.2. (Closed Partial Order) *We say that a partial order p is closed if there exists no other partial order p' with $p \trianglelefteq p'$ s.t. $support(p) = support(p')$.*

A closed partial order is the most specific one among all those posets compatible with the same set of input sequences, so they are the most informative ones. Actually, we can prove the following result.

Lemma 5.2. *Given a valid pair (S, T), a partial order obtained by matching all the path preserving positions of sequences in S is a closed partial order.*

Proof. It follows from lemma 5.1: part (1) ensures that the generated partial order whose maximal paths are in S is compatible with all transactions T; part (2) of the same lemma ensures that this partial order cannot contain other paths not included in S, otherwise it is not compatible with T. Then, by matching all the path preserving positions from sequences in S we get a partial order p such that for no other partial order p' we have that $p \trianglelefteq p'$ and p' compatible with T. Thus, p is closed.

Note that closed partial orders can be obtained only by means of a set of closed sequences, that is, from the set S of each valid pair (S, T); otherwise the partial orders would not be closed. The set of closed partial orders describe the input sequential data $\mathcal{D}$ in the most specific way. In terms of category theory, this process can be formalized as a coproduct transformation when not considering repetition of items in the input sequences (already proved in [8])

In the practice, this equivalence between groups of closed sequences and closed partials orders represents an important algorithmic simplification, since algorithms such as BIDE or CloSpan or TSP are now able to efficiently transform their patterns into closed partial orders and we do not need to mine them directly from the data. Note that if a minimum support condition is specified over the closed sequences mined by those algorithms, then the generated closed partial orders will be also over that minimum support.

6 Further Discussions

In this section we would like to note first that the contribution presented here also works when considering simultaneity condition of input sequences, that is, when having input sequences of itemsets, as in example of figure 7. Notice that algorithms such as CloSpan, BIDE or TSP are already able to mine closed sequential patterns when having sequences of itemsets, and grouping those sequences in valid pairs (S, T) can be done with the described algorithm.

Seq id	**Input sequences**
t_1	$\langle(AF)(D)(E)(A)\rangle$
t_2	$\langle(E)(ABF)(G)(BDE)\rangle$
t_3	$\langle(E)(A)(B)(G)\rangle$

Figure 7: Collection of data $\mathcal{D}$

The way of matching positions of sequences $s \in S$ for each (S,T) will be done as usual, that is, according to definition 5.1. This definition takes care of not adding new paths in the generated closed partial order: because of that, the set of items of two different positions must coincide exactly to be matched. The results obtained from the example of figure 7 are shown in figure 8 and figure 9.

T	S
$\{1,2,3\}$	$\{\langle(E)(A)\rangle\}$
$\{2,3\}$	$\{\langle(E)(A)(B)\rangle, \langle(E)(A)(G)\rangle, \langle(E)(B)(G)\rangle\}$
$\{1,2\}$	$\{\langle(AF)(D)\rangle, \langle(AF)(E)\rangle, \langle(E)(A)\rangle\}$
$\{1\}$	$\{\langle(AF)(D)(E)(A)\rangle\}$
$\{2\}$	$\{\langle(E)(ABF)(G)(BDE)\rangle\}$
$\{3\}$	$\{\langle(E)(A)(B)(G)\rangle\}$

Figure 8: Groups of closed sequences occurring together

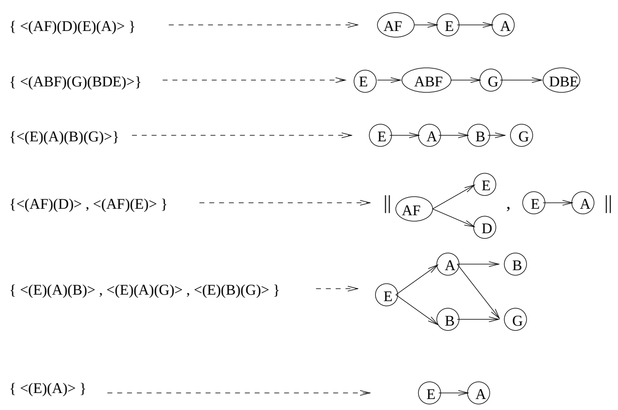

Figure 9: Partial orders from (S,T)

A different observation is that instead of generating all the closed partial orders out of the closed sequences, we may want to generate only those partial orders whose support is over a maximum threshold. To perform this transformation is also easy since we just need to select those valid pairs (S,T) s.t. the number of transactions in T is over this maximum value. We can play with other parameters, such as the length or the structure of the partial order.

Finally, we want to raise the discussion of how to represent the obtained partial orders. A first approach is to use adjacency matrices and consider that each different node is an entry of the matrix. It is easy to transform a group of sequences S and the list of overlapping positions given by algorithm 2 into an adjacency matrix. Other more visual solutions would involve graphical representations of directed acyclic graphs, which can certainly ease the interpretation of the partial order made by the user. We are currently working towards a visualization of the final partial orders with GraphML.

7 Experimental Results

We evaluate our approach by performing experiments on different discrete sequential databases: a first database of 1000 transactions of synthetic data (we used a context-free grammar as generative model for sequences of words); a second database of 607 transactions corresponding to the command history of a unix computer user (downladed from the UCI repository[1]); and a third database corresponding to the first chapter of the book "1984" by George Orwell, where each different word is considered a different item and each sentence an input sequence. We are aware that these two real databases are quite small; but due to our lack of a large real dataset, we decided to perform here just a preliminary experimentation as an overview of the first results.

Our first goal is to evaluate the number of total closed partial orders in comparison to closed sequential patterns, and also, to analyze the quality of those partial orders. The process is first mining the closed sequential patterns over a certain minimum threshold named σ (we can use any existing algorithm), and then, organizing the closed sequences into valid pairs, as described in section 4. We also want to compare the performance of these two phases.

A first set of numerical comparisons obtained with the synthetic data are shown in table 1. We observe that the number of closed partial orders is always less than or equal to the number of closed sequential patterns; moreover, as we decrease the minimum support and we get more frequent sequences, the number of closed partial orders gets considerabley smaller. Actually, we generated a first set of only 10000 synthetic transactions, to show that, even when the number of frequent sets and closed sequences is larger than the number of transactions, the number of closed posets never beats this limit.

σ	**Frequent Seqs.**	**Closed Seq.**	**Closed Posets**
200	31	26	26
100	114	92	92
50	520	401	92
40	850	645	224
30	1467	1108	389

Table 1: Counting of patterns for the synthetic data

The same results are shown in figure 2 using the unix command data. Here, we forced the minimum support to very low values, in order to evaluate the number

[1] http://kdd.ics.uci.edu/summary.data.type.html

of closed sequences with respect to the total posets. We still have that the number of closed partial orders is less than the number of closed sequences; however, we see this number is sometimes larger than the 607 original transactions, making the final patterns less manageable. Some real closed partial orders extracted from this database are shown in next figure 10.

σ	**Frequent Seqs.**	**Closed Seq.**	**Closed Posets**
30	277	259	259
20	1111	913	885
10	23460	1890	1565
5	37898	4778	3779

Table 2: Counting of patterns for the Unix command database

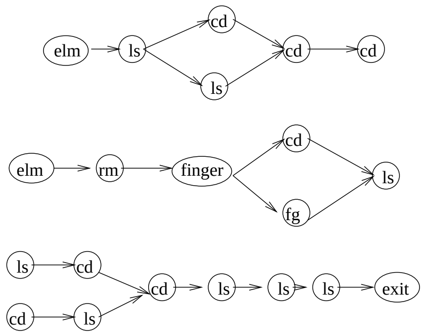

Figure 10: Some partial orders obtained from the Unix command database

In case we are dealing with unix user data, the frequent closed partial orders may be later used as the normal user profile for the intrusion detection systems (such as proposed in [9]). However, these closed posets may have other utilities in the field of knowledge discovery depending on the context. For example, in case of having a text (such as our second database with a text written by George Orwell), the closed partial orders can be used to classify subsequent texts according to a list of authors. In the first chapter of "1984" we have a total of 340 input sequences. We experimented with different minimum supports, and some of the obtained closed posets are shown in figure 11.

As mentioned, we can divide our discovery process in two phases: a first one of finding the closed sequences, and a second phase of obtaining valid pairs and closed partial orders from there. We observed that the first phase is the most I/O intensive, since it requires to examine a combinatorial number of patterns. On the other hand, the second phase is not an intensive step: the input to be examined is only the set of frequent closed

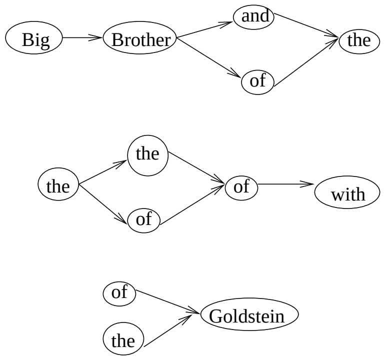

Figure 11: Some partial orders obtained from the "1984" database

sequences, and we do not require to combine those patterns, just organizing them in the conceptual struture; moreover, the operations performed to get such lattice are simply standard operations of sets, such as inclusion, intersection and so on. With small datasets, as the ones described here, this organization takes only a few seconds. We must point out though, that this second phase can be more costly when considering a very large database: in this case the tid lists are larger and the comparisons between these lists can be more expensive; however, we evaluated with synthetic data that, regardless the length of the database, the cost of organizing closed sequences is still insignificant compared to the first phase.

With these preliminary experiments we wanted to show that the main profit of our proposal is the possibility of generating classical partial orders out of closed sequences, without dealing direclty with the input data.

8 Related Work

The importance of mining partial order structures from sequential data was first introduced in [11]. There, the authors start with a slightly different data model described as a long sequence of events; in the practice, this long sequence can be divided in several sliding windows, so, coinciding with the transactional model presented here.

The basic problem defined in [11] is to find frequent episodes, i.e. collections of events occurring frequently together in the input sequence. Episodes are formalized as acyclic directed graphs, so, it is equivalent to the one presented in section 5 of this paper. Episodes can be classified into: serial episodes (total orders), parallel episodes (trivial orders), and finally, hybrid episodes (indeed, general partial orders).

The work in [11] discusses different algorithmic approaches for the discovery of serial, parallel and hybrid episodes. In particular, the popular approach called Winepi is intended to look for frequent episodes in a Apriori fashion by sliding a window of fixed width along the event sequence: i.e., a complete pass along the data is used to compute the support of current episode candidates and, after each pass, new larger episodes are generated as long as the antimonotonicity property of support keeps them alive. So, at the end of the process Winepi has discovered all the frequent episodes of any kind fitting in the window.

This mentioned algorithmic approach performs two complex operations: first, generating new candidate episodes out of the smaller ones, and second, recognizing episodes in the sequence to update the support. In case of mining dense data, and specially, when dealing with hybrid episodes (i.e., general partial orders) or with non-injective episodes (those having repeated items), the algorithm incurs in a substantial runtime overhead (see [11] for more details). Apart from this algorithmic overhead, the number of the final discovered episodes is quite large and many of these episodes are not the most specific ones and they could be considered redundant: e.g. many of the final parallel episodes may be less informative than some of the serial episodes, and also, many serial episodes may be less informative than the hybrid episodes.

The proposals suggested in this paper reduce the discovery of episodes to the most specific ones, i.e. those which are closed. In this sense, it represents a semantic advance of hybrid episodes, since we just consider the most informative partial orders to summarize the database. Moreover, we show that it is not necessary to mine these structures directly from the data (as it is done by the Winepi approach), but just post-processing the closed sequential patterns. Operations done by algorithms such as CloSpan or BIDE are less expensive than Winepi since they just compare plain sequences, so, the final process is less costly.

Alternatively, the work in [11] proposes also another algorithmic approach called Minepi. In this case the mined episodes are unbounded in length, and for the moment, it is not clear how our proposal can improve semantically these episodes. Another work worth mentioning is [10]: it presents a method based on viewing a partial order as a generative model for a set of sequences and it applies different mixture model techniques. The final partial orders are not necessarily closed and so, they could be redundant; besides, they restrict the attention to a subset of partial orders called series-parallel partial orders (such as series-parallel digraphs) to avoid computational problems. Note that here we do not restrict in any sense the form of the final closed posets or the repetition of items.

Other works worth mentioning are [2] or [4, 14], but they deal with serial episodes more than hybrid structures as we manage here. Moreover, the difference with respect the current proposal, is that we are considering the discovery of episodes as a post-processing step of closed sequences, more than a direct discovery from the original data.

9 Conclusions

We have presented the notion of closed partial orders compatible with input sequences of itemsets. We show that this is a general data model that can be adapted to long sequences of events as well. By definition, these orders are the most informative ones for a set of maximal transactions and they provide a more compact overview of the input data. As a main contribution, we show that these closed partial orders can be derived simply from the closed set of sequences mined by existing algorithms.

In the practice, this transformation implies that going beyond closed partial orders once we have the closed sequential partterns is not costly. So, current algorithms can efficiently transform the discovered patterns into closed partial orders; thus, avoiding the complexity of mining these structures directly from the data. First experiments show that postprocessing closed sequences is not a costly phase, so that the real profit of the proposal is in the ability to generate closed partial orders without accessing the input data, more than the reduction in the number of patterns.

Yet there is still the work of formalizing the matching process of closed sequences into a partial order. We mentioned that in case of not having repetition of items in the input sequences, this process can be formalized by means of coproduct operations of category theory (as it shows [8], where also the necessary closure system of partial orders is characterized). The next step is to do this formalization for the general data model, that is, when having repetition of items and simultaneity. We are currently working towards this characterization by means of colimit operations of category theory.

Other further work includes the study of implications or association rules of partial orders. Given that implications among closed sequential patterns are already formalized, it is reasonable to go further and find a similar characterization for these closed partial orders.

Acknowledgements: The author thanks José L. Balcázar for his useful discussions and comments, and also Pablo Díaz-López for helping with the implementations and experiments.

References

[1] R. Agrawal and R. Srikant. Mining sequential patterns. In *Proceedings of the 11th Int. Conference on Data Engineering*, pages 3–14, 1995.

[2] M.J. Atallah, R. Gwadera, and W. Szpankowski. Detection of significant sets of episodes in event sequences. In *Proceedings of the 4th International Conference on Data Mining*, pages 3–10, 2004.

[3] J.L. Balcázar and G.C. Garriga. On Horn axiomatizations for sequential data. In *Proceedings of the 10th Int. Conference on Database Theory*, pages 215–229, 2005.

[4] G. Das, R. Fleischer, L. Gasieniec, D. Gunopulos, and J. Kärkkäinen. Episode matching. In *Proceedings of the 8th Annual Symposium on Combinatorial Pattern Matching*, pages 12–27, 1997.

[5] B.A. Davey and H.A. Priestly. *Introduction to Lattices and Order*. Cambridge, 2002.

[6] B. Ganter and R. Wille. *Formal Concept Analysis. Mathematical Foundations*. Springer, 1998.

[7] G.C. Garriga. Towards a formal framework for mining general patterns from structured data. In *2nd Int. KDD Workshop on MRDM'03*, pages 215–229, 2003.

[8] G.C. Garriga and J.L. Balcázar. Coproduct transformations on lattices of closed partial orders. In *Proceedings of 2nd. Int. Conference on Graph Transformation*, pages 336–351, 2004.

[9] W. Lee, S.J. Stolfo, and K. Mok. A data mining framework for building intrusion detection models. In *Proceedings of the IEEE Symposium on Security and Privacy*, pages 120–132, 1999.

[10] H. Mannila and C. Meek. Global partial orders from sequential data. In *Proceedings of the 6th Int. Conf. on Knowledge Discovery in Databases*, pages 161–168, 2000.

[11] H. Mannila, H. Toivonen, and A.I. Verkamo. Discovering frequent episodes in sequences. *Data Mining and Knowledge Discovery*, 1(3):259–289, 1997.

[12] N. Pasquier, Y. Bastide, R. Taouil L., and Lakhal. Closed set based discovery of small covers for association rules. In *Proceedings of the 15th Int. Conference on Advanced Databases*, pages 361–381, 1999.

[13] J. Pei, J. Han, B. Mortazavi-Asl, H. Pinto, Q. Chen, U. Dayal, and M. Hsu. PrefixSpan: mining sequential patterns by prefixprojected growth. In *Proceedings of the 17th Int. Conference on Data Engineering*, pages 215–224, 2001.

[14] Z. Tronicek. Episode matching. In *Combinatorial Pattern Matching*, pages 143–146, 2001.

[15] P. Tzvetkov, X. Yan, and J. Han. TSP: Mining top-k closed sequential patterns. In *Proceedints of the 3rd IEEE International Conference on Data Mining*, pages 347–358, 2003.

[16] J. Wang and J. Han. BIDE: Efficient mining of frequent closed sequences. In *Proceedings of the 19th Int. Conference on Data Engineering*, pages 79–90, 2003.

[17] X. Yan, J. Han, and R. Afshar. CloSpan: Mining closed sequential patterns in large datasets. In *Proceedings of the Int. Conference SIAM Data Mining*, pages 166–177, 2003.

[18] M. Zaki. Generating non-redundant association rules. In *Proceedings of the 6th Int. Conference on Knowledge Discovery and Data Mining*, pages 34–43, 2000.

[19] M. Zaki. SPADE: An efficient algorithm for mining frequent sequences. *Machine Learning Journal, special issue on Unsupervised Learning*, 42(1/2):31–60, 2001.