

Efficient Satisfiability-Based Approach for Mining Partially-Ordered Sequential Rules

Abdelhamid Boudane(✉), Lamia Sadeg, and Mohamed Maiza

LMTO/UER-ISI, Ecole Militaire Polytechnique, Algiers, Algeria
{abdelhamid.boudane,lamia.sadeg,mohamed.maiza}@emp.mdn.dz

Abstract. Sequential rule mining is a key task in data mining that aims to discover temporal relationships between itemsets within sequence data. Among the different types of sequential rules, partially ordered sequential rules (*POSRules*) have gained attention due to their ability to capture precedence relationships without enforcing strict ordering within itemsets. However, existing approaches for mining *POSRules* often suffer from combinatorial explosion, high memory consumption, and limited flexibility in handling additional constraints.

In this paper, we propose a novel satisfiability (SAT)-based approach for extracting *POSRules*. The proposed method encodes the mining problem as a propositional formula whose models correspond to valid rules. To improve scalability, we introduce a decomposition strategy that partitions the original problem into smaller and more manageable subproblems, whose combined solutions yield the complete set of rules.

An extensive experimental evaluation conducted on several real-world datasets demonstrates that the proposed approach outperforms state-of-the-art algorithms in terms of efficiency and scalability. These results highlight the potential of declarative methods as a powerful alternative for sequential rule mining.

Keywords: Data mining · Sequential rules · Satisfiability · Propositional logic

1 Introduction

Data mining is a fundamental field that focuses on extracting hidden patterns and relationships from data. These relationships provide useful insights that help better understand the behavior of the studied phenomena. The type of the extracted knowledge depends on both the nature of the data and the objectives of the analysis. Among the various data mining tasks, sequential pattern mining plays an important role when dealing with ordered data. It has shown strong effectiveness in many application domains. For example, in e-commerce, it helps model customer purchasing behavior and improve recommendation systems by analyzing frequent purchase and click sequences [5,8]. In healthcare, it is used to

© The Author(s), under exclusive license to Springer Nature Switzerland AG 2027
O. Gervasi et al. (Eds.): ICCSA 2026, LNCS 16769, pp. 129–145, 2027.
https://doi.org/10.1007/978-3-032-30494-0_9

discover temporal relationships in clinical data, supporting outcome prediction and decision-making [17]. These examples illustrate the ability of sequential pattern mining to capture meaningful temporal dependencies in complex sequence data.

Within different types of patterns in sequence data, sequential rules are of particular interest. They are expressed as implications of the form $X \rightarrow Y$ and can be used to predict future events in sequences. The literature distinguishes two main categories of sequential rules. The first category includes rules where both the antecedent X and the consequent Y are sequential patterns [20]. The second category consists of rules defined between two unordered sets of items [9,10,12].

In this paper, we focus on the second category, namely partially ordered sequential rules ($POSRules$) [10], which extend association rule mining by discovering relationships of the form $X \rightarrow Y$, where the items in X precede those in Y within sequences. In this setting, no ordering is imposed among the items within X or within Y; however, all items in X must occur before those in Y, hence the term partially ordered. The relevance of such rules is typically evaluated using support and confidence measures, which respectively quantify the frequency of a rule in the dataset and the strength of the implication between its antecedent and consequent.

Several algorithms have been developed to efficiently mine $POSRules$. Early methods, such as CMDeo [9], use an Apriori-based level-wise strategy that generates and tests candidate rules. While effective on small datasets, this approach does not scale well due to the large number of candidates. To overcome this limitation, CMRules [10] first extracts frequent itemsets and generates association rules, which are then filtered to meet sequential constraints. This reduces the search space compared to CMDeo, but its performance still depends on the number of generated rules. More recent approaches use pattern-growth techniques. For example, RuleGrowth [12] avoids candidate generation by directly extending rules on both sides, improving efficiency and scalability. Similarly, ERMiner [11] further improves performance using equivalence classes and advanced pruning, achieving faster runtime at the cost of higher memory use.

Despite these advances, existing approaches for mining $POSRules$ still face major challenges. They often suffer from combinatorial explosion, high memory requirements, and limited flexibility in handling additional constraints. These limitations motivate the exploration of alternative paradigms that can improve both efficiency and expressiveness in sequential rule mining.

Declarative approaches, such as satisfiability (SAT) and constraint programming, have been successfully applied to many data mining tasks [7], including association rule mining [3,4,14], community detection [16], and frequent sequence mining [1]. However, to the best of our knowledge, they have not yet been used for extracting partially ordered sequential rules. In this paper, we propose a SAT-based approach for mining $POSRules$. The method encodes the problem as a propositional formula, where each model corresponds to a valid rule. It also introduces a decomposition strategy that breaks the problem into

smaller subproblems, whose combined solutions form the complete set of rules. An extensive experimental evaluation on real-world datasets demonstrates that the proposed approach outperforms state-of-the-art methods in both efficiency and scalability.

2 Preliminaries and Problem Statements

2.1 Partially Ordered Sequential Rules Mining Problem

Let Σ be a finite non empty set of symbols, called *items*. An *itemset* X over Σ is defined as a subset of Σ, i.e., $X \subseteq \Sigma$. From now, we use the capital letters X, Y, K, etc. to represent itemsets and lowercase letters a, b, c, etc. to denote the elements of Σ.

A *sequence* S_i is a pair (i, S) where i is the sequence identifier and S is an ordered list of itemsets , i.e., $S = \{X_1, ..., X_n\}$. A *sequence database* $\mathcal{SD}$ is a finite non empty set of sequences where each sequence identifier refers to a unique list of itemsets.

An itemset X is covered by a sequence S if each item appearing in X appears in S. Then, the *cover* of an itemset X in a sequence database $\mathcal{SD}$, denoted $Cov(X, \mathcal{SD})$, is defined as $\{i \in \mathbb{N} \mid (i, S) \in \mathcal{SD}\ and\ X \subseteq \bigcup_{I \in S} I\}$.

The *support* of X in $\mathcal{SD}$, denoted $Supp(X, \mathcal{SD})$, corresponds to the cardinality of $Cov(X, \mathcal{SD})$, i.e., $Supp(X, \mathcal{SD}) = \mid Cov(X, \mathcal{SD}) \mid$.

A *partially-ordered sequential rule* ($POSRule$), is a pattern of the form $X \rightarrow Y$ where X and Y are two disjoint itemsets called respectively the antecedent and the consequence of the rule. the relevance of such rules is defined using two well known predicates namely support and confidence. We say that a POSRule $X \rightarrow Y$ is coverd by a sequence $S = \{I_1, ..., I_n\}$ and we note $(X \rightarrow Y) \sqsubseteq S$ iff $\exists\ i,\ j,\ k,\ l \in \mathbb{N} \mid\ 1 \leq i \leq j < k \leq l \leq n$ and $X \subseteq \{\bigcup_{m=i}^{m=j} I_m\}$ and $Y \subseteq \{\bigcup_{m=k}^{m=l} I_m\}$. The constraint $j < k$ represents the fact that the items of Y must appear after the items of X while there is no order on the elements of X and Y.

The *cover* of a $POSRule\ X \rightarrow Y$ in a sequential database $\mathcal{SD}$, is defined as $\mathcal{C}ov(X \rightarrow Y, \mathcal{SD}) = \{i \in \mathbb{N} \mid (i, S) \in \mathcal{SD}\ and\ (X \rightarrow Y) \sqsubseteq S_i\}$. Then, the *support* of a $POSRule\ X \rightarrow Y$ in $\mathcal{SD}$, is defined as $Supp(X \rightarrow Y, \mathcal{SD}) = \frac{|\mathcal{C}ov(X \rightarrow Y, \mathcal{SD})|}{|\mathcal{SD}|}$. Such support measure determines the frequency of a candidate $POSRule$ in a given sequential database $\mathcal{SD}$.

The *confidence* of $X \rightarrow Y$ in $\mathcal{S}$, defined as $\mathcal{C}onf(X \rightarrow Y, \mathcal{S}) = \frac{Supp(X \rightarrow Y, \mathcal{SD})}{Supp(X, \mathcal{SD})}$, represents the conditional probability of the appearance of Y after a given X in $\mathcal{SD}$. When there is no ambiguity, we omit to mention the sequential database $\mathcal{SD}$, and we simply note $Supp(X \rightarrow Y)$ and $\mathcal{C}onf(X \rightarrow Y)$. Note that support (respectively confidence) measures can be expressed as a percentage by multiplying it by 100.

This work addresses the problem of mining partially ordered sequential rules, formalized as follows:

Definition 1 (Problem of mining POSRules). *Given a sequence database $\mathcal{SD}$, a minimum support threshold λ and a minimum confidence threshold θ, the problem of mining POSRules consists in finding all the pairs of itemsets (X,Y) so that the following constraints are satisfied:*

- $X \cap Y = \emptyset$.
- $Supp(X \rightarrow Y) \geq \lambda$.
- $\mathcal{C}onf(X \rightarrow Y) \geq \theta$

Formally, the set of partially ordered rules is defined as follows:

$$POSRules(\mathcal{SD}, \lambda, \theta) = \{X \rightarrow Y \mid X, Y \subseteq \Sigma,\ Supp(X \rightarrow Y) \geq \lambda, \\ and\ \mathcal{C}onf(X \rightarrow Y) \geq \theta\}$$

Example 1. Consider the sequence database $\mathcal{SD}$, represented in Table 1, containing four sequences over the set of items $\Sigma = \{a, b, c, d, e, f, g\}$. For a minimum support threshold $\lambda = 0.5$ and a minimum confidence threshold $\theta = 0.7$ there are five partially ordered rules illustrated in Table 2. For example, for the rule $\{b\} \rightarrow \{f\}$ we have $\mathcal{C}ov(\{b\} \rightarrow \{f\}, \mathcal{SD}) = \{1, 2, 4\}$ and $\mathcal{C}ov(\{b\}, \mathcal{SD}) = \{1, 2, 3, 4\}$. Therefore, $Supp(\{b\} \rightarrow \{f\}, \mathcal{S}) = 3/4$ and $\mathcal{C}onf(\{b\} \rightarrow \{f\}, \mathcal{S}) = 3/4$.

Table 1. A Database $\mathcal{SD}$

S_{id}	*Sequences*
1	ab, g, f, c, e
2	de, bc, a, f, b
3	a, g, b, ac
4	b, fe

Table 2. POSRules

Rules	*Supp*	$\mathcal{C}onf$
$\{b\} \rightarrow \{f\}$	3/4	3/4
$\{g\} \rightarrow \{c\}$	2/4	2/2
$\{a, g\} \rightarrow \{c\}$	2/4	2/2
$\{a, b, g\} \rightarrow \{c\}$	2/4	2/2
$\{b, g\} \rightarrow \{c\}$	2/4	2/2

2.2 Satisfiability Problem in Propositional Logic

We introduce in the following the fundamental concepts of propositional logic. Indeed, We define at first the syntax and the semantics of propositional logic and we present at second the satisfiability problem in such logic.
A propositional variable x is a Boolean input variable of a truth function that takes values in {0,1}. 0 and 1 are used to denote $False$ ($\bot$) and $True$ ($\top$) values. Let $\mathcal{B}$ be the set of Boolean variables. The set of propositional formulas denoted by $\mathcal{F}$ is defined from $\mathcal{B}$, $\bot$, $\top$ and the logical connectives ($\neg$, $\wedge$, $\vee$, $\rightarrow$, and $\leftrightarrow$) so that:

- A propositional variable $x \in \mathcal{B}$, $\bot$, and $\top$ are formulas;
- If ϕ and ψ are propositional formulas, then $\neg\phi$, $\phi \wedge \psi$, $\phi \vee \psi$, $\phi \rightarrow \psi$ and $\phi \leftrightarrow \psi$ are also propositional formulas.

We use $\mathcal{V}(\phi)$ to denote the set of Boolean variables appearing in formula ϕ.
Basically, logical reasoning solvers take input formulas in normal forms like

conjunctive normal form (CNF). A CNF is a conjunction of clauses. A clause is a disjunction of literals where a literal is a positive propositional variable x or its negation $\neg x$.
An interpretation $\mathcal{I}$ of a propositional formula ϕ is a function from $\mathcal{V}(\phi)$ to $\{0,1\}$ which associates to each Boolean variable $x \in \mathcal{V}(\phi)$ a value $\mathcal{I}(x) \in \{0,1\}$. An interpretation $\mathcal{I}$ that satisfies a formula ϕ ($\mathcal{I}(\phi) = 1$) is called model of ϕ.
Given a CNF formula ϕ, The Satisfiability (SAT) problem consists on deciding if ϕ admits a model or not. SAT is a fundamental problem in computational complexity theory because it was the first problem to be proved NP-Complete [6]. Its worth noting that the mostly used satisfiability solvers takes as input a CNF formula and any propositional formula could be transformed to an equisatisfiable CNF formula in polynomial time and space [19].

3 SAT Encoding of Mining Partially Ordered Sequential Rules

In this section, we propose a SAT-based encoding for the problem of mining partially ordered sequential rules (*POSRules*). The approach consists in representing the mining task as a propositional formula in conjunctive normal form (CNF), such that each satisfying assignment corresponds to a valid *POSRule*. This encoding relies on the definition of propositional variables along with a set of constraints that precisely capture the semantics of the problem. We begin by introducing the variables used to represent the different components and positions of a sequential rule. We then present the constraints required to ensure a correct and complete modeling of the *POSRules* mining problem.

3.1 Variables Definition

Given a set Σ of n items, a sequential database $\mathcal{SD} = \{(1, S_1), ..., (m, S_m)\}$ where $\forall i \in \{1, ..., m\}, S_i = \{I_1 \subseteq \Sigma, ..., I_k \subseteq \Sigma\}$, a minimum support threshold λ and a minimum confidence threshold θ. We define two sets of Boolean variables $\{x_a \mid a \in \Sigma\}$ and $\{y_a \mid a \in \Sigma\}$ to represent the items of a candidate sequential rule $X \rightarrow Y$. In fact, true variables from the two sets, in a given model $\mathcal{M}$ of the SAT encoding, represent the *POSRule* $X = \{x_{a \in \Sigma},\ x_a = 1\} \rightarrow Y = \{y_{a \in \Sigma},\ y_a = 1\}$.

We associate also to each sequence identifier i two propositional variables p_i and q_i respectively. This variables allows us to model the fact that a sequence (i, S) in $\mathcal{SD}$ covers the itemset X and the candidate rule $X \rightarrow Y$ respectively. Formally, in a model $\mathcal{M}$ of the considered encoding, the two sets $\{i \in \{1, ..., m\} \mid \mathcal{M}(p_i) = 1\}$ and$\{\{i \in \{1, ..., m\} \mid \mathcal{M}(q_i) = 1\}\}$ represent respectively $Cov(X, \mathcal{SD})$ and $Cov(X \rightarrow Y, \mathcal{SD})$.

For example, the corresponding Boolean variables of the dataset $\mathcal{SD}$ presented in Table 1 are $X = \{x_a, x_b, x_c, x_d, x_e, x_f, x_g\}$, $Y = \{y_a, y_b, y_c, y_d, y_e, y_f, y_g\}$, $Cov(X, \mathcal{SD}) = \{p_1, p_2, p_3, p_4\}$, and $Cov(X \rightarrow Y, \mathcal{SD}) = \{q_1, q_2, q_3, q_4\}$.

3.2 Constraints Formulation

We now introduce the set of constraints used to model the *POSRules* extraction problem. Recall that the validation of a candidate rule $X \rightarrow Y$ relies on the computation of support and confidence measures. In the proposed encoding, these measures require the definition of constraints that capture both the cover of the antecedent, denoted $\mathcal{C}ov(X, \mathcal{SD})$, and the cover of the rule, denoted $\mathcal{C}ov(X \rightarrow Y, \mathcal{SD})$. To this end, we encode the occurrence of the candidate rule $X \rightarrow Y$ and its antecedent X in each sequence S_i using Boolean variables q_i and p_i, respectively. These variables are then used within linear constraints to express the support and confidence conditions.

We now present the different constraints that model the problem of mining *POSRules*.

1. *The two sides of the rule are not empty (i.e. $X \neq \emptyset$ and $Y \neq \emptyset$):* this constraint is expressed by two clauses (see formula (1)) that enforce the candidate rule to have at least one item in X and at least one item in Y. It avoids candidate rules that doesn't have conditions or consequences like ($\rightarrow y_b$ *or* $x_a \rightarrow$).

$$(\bigvee_{a \in \Sigma} x_a) \wedge (\bigvee_{a \in \Sigma} y_a) \tag{1}$$

2. *An item can appear in at most one side of a candidate rule (i.e. $X \cap Y = \emptyset$):* This constraint (see Formula (2)) ensures that no item is shared between the antecedent X and the consequent Y, thereby eliminating invalid rules such as $ac \rightarrow ab$. More precisely, for each item $a \in \Sigma$, a clause is introduced to enforce that a cannot simultaneously belong to both X and Y.

$$\bigwedge_{a \in \Sigma} (\neg x_a \vee \neg y_a) \tag{2}$$

3. *A sequence S_i covers the left side of a candidate rule $X \rightarrow Y$:* This constraint captures whether the itemset X appears in the sequence S_i. Specifically, Formula (3) enforces the variable p_i to be false if and only if at least one item of X does not appear in S_i.

$$\neg p_i \leftrightarrow \bigvee_{a \notin s_i} x_a \tag{3}$$

4. *The itemset Y appears after the itemset X in a sequence S_i:* To model this constraint, an additional variable z_i is introduced to ensure that all items in X occur before those in Y within S_i. This condition is captured by Formula (4), which enforces that each item $y \in Y$ appears after the first occurrence of every item $x \in X$.
   In other words, if there exists an item $a \in X$ whose first occurrence is after the last occurrence of an item $b \in Y$, then the candidate rule $X \rightarrow Y$ violates the ordering constraint in sequence S_i, and the variable z_i is set to false.

$$\forall S_i \in \mathcal{SD}, \bigwedge_{a \in s_i,\ b \in s_i} z_i \rightarrow (x_a \rightarrow \neg y_b); \quad FOcc(a, s_i) \geq LOcc(b, s_i). \tag{4}$$

Note that $FOcc(a, s_i)$ (respectively $LOcc(b, s_i)$) is a function that returns the position of the first (respectively the last) occurrence of the item a (respectively b) in a sequence S_i. Moreover, formula (4) can be simply rewritten as a conjunction of clauses using the equivalence $A \rightarrow B \equiv \neg A \vee B$ as follows:

$$\forall S_i \in \mathcal{SD}, \bigwedge_{a \in s_i,\ b \in s_i} (\neg z_i \vee \neg x_a \vee \neg y_b); \quad FOcc(a, s_i) \geq LOcc(b, s_i).$$

5. *A sequence S_i covers the candidate rule $X \rightarrow Y$:* This constraint uses the variable q_i to represent whether a candidate rule $X \rightarrow Y$ is covered by a sequence S_i. For this to hold, both X and Y must appear in S_i, and the items of X must precede those of Y. If any of these conditions is not satisfied, the variable q_i is set to false. Formula (5) encodes this constraint by combining the variable p_i, which represents the cover of X, and the variable z_i, which enforces the ordering constraint. In addition, the subformula $\bigvee_{a \notin S_i} y_a$ is used to capture the cover of Y.

$$\neg q_i \leftrightarrow \neg p_i \vee \bigvee_{a \notin s_i} y_a \vee \neg z_i \tag{5}$$

6. *Minimum support:* This constraint encodes the fact that a candidate rule $X \rightarrow Y$ should have a minimum support λ in the database $\mathcal{SD}$ to be a valid *POSRule*. In fact, it should be covered by at least λ sequences. Such cover tests are captured by variables q_i from formula (5). Then, the sum of true q_i variables gives us the support value of the candidate rule $X \rightarrow Y$ as illustrated in formula (6).

$$\sum_{i=1}^{i=m} q_i \geq \lambda \tag{6}$$

7. *Minimum confidence:* To express this constraint, it is necessary to compute both the support of the itemset X and the support of the candidate rule $X \rightarrow Y$. These values are obtained by summing the variables p_i and q_i, which are defined by the cover constraints in Formulas (3) and (5), respectively. The minimum confidence constraint is then modeled as a 0/1 linear inequality, as shown in Formula (7).

$$\frac{\sum_{i=1}^{i=m} q_i}{\sum_{i=1}^{i=m} p_i} \geq \theta \tag{7}$$

By considering that the confidence threshold θ takes values between 0 and 1, we can rewrite the constraint 7 as follows:

$$100 * \sum_{i=1}^{i=m} q_i - 100 * \theta * \sum_{i=1}^{i=m} p_i \geq 0 \tag{8}$$

It is worth noting that the two formulas (6) and (8) are called cardinality and pseudo Boolean constraints. They could be translated efficiently into CNF formulas using one of the literature encodings ([2,18]).

We use $\Phi_{POSRules}(\mathcal{SD}, \lambda, \theta)$ to denote the encoding corresponding to the conjunction of the propositional formulas (1), (2), (3), (4), (5), (6), and (8).

Proposition 1. *Let $\mathcal{SD}$ a sequential database, λ a minimum support threshold, θ a minimum confidence threshold and $\mathcal{I}$ an interpretation of $\Phi_{POSRules}(\mathcal{SD}, \lambda, \theta)$. Then, $\mathcal{I}$ is a model of $\Phi_{POSRules}(\mathcal{SD}, \lambda, \theta)$ iff $X = \{a \in \Sigma | \mathcal{I}(x_a) = 1\} \rightarrow Y = \{b \in \Sigma | \mathcal{I}(y_b) = 1\}$ is a POSRule with $\mathcal{C}ov(X \rightarrow Y, \mathcal{SD}) = \{i \in \{1..m\} | \mathcal{I}(q_i) = 1\}$.*

Example 2. Let us consider the sequencial database of Table 1. The following formula encodes the problem of mining Partially Ordered Rules with a minimum support threshold $\lambda = 0.5$ (at least 2 sequences covers the rule) and a minimum confidence threshold $\theta = 0.5$:

$$
\begin{aligned}
&(x_a \vee x_b \vee x_c \vee x_d \vee x_e \vee x_f \vee x_g) \wedge (y_a \vee y_b \vee y_c \vee y_d \vee y_e \vee y_f \vee y_g) \wedge \\
&(\neg x_a \vee \neg y_a) \wedge (\neg x_b \vee \neg y_b) \wedge (\neg x_c \vee \neg y_c) \wedge (\neg x_d \vee \neg y_d) \wedge (\neg x_e \vee \neg y_e) \wedge \\
&(\neg x_f \vee \neg y_f) \wedge (\neg x_g \vee \neg y_g) \wedge \\
&(\neg p_1 \leftrightarrow x_d) \wedge \\
&(\neg p_2 \leftrightarrow x_g) \wedge \\
&(\neg p_3 \leftrightarrow x_d \vee x_e \vee x_f) \wedge \\
&(\neg p_4 \leftrightarrow x_a \vee x_c \vee x_d \vee x_g) \wedge \\
&(z_1 \rightarrow (x_a \rightarrow \neg y_b)) \wedge (z_1 \rightarrow (x_b \rightarrow \neg y_a)) \wedge (z_1 \rightarrow (x_c \rightarrow \neg y_a)) \wedge \\
&(z_1 \rightarrow (x_c \rightarrow \neg y_b)) \wedge (z_1 \rightarrow (x_c \rightarrow \neg y_f)) \wedge (z_1 \rightarrow (x_c \rightarrow \neg y_g)) \wedge \\
&(z_1 \rightarrow (x_e \rightarrow \neg y_a)) \wedge (z_1 \rightarrow (x_e \rightarrow \neg y_b)) \wedge (z_1 \rightarrow (x_e \rightarrow \neg y_c)) \wedge \\
&(z_1 \rightarrow (x_e \rightarrow \neg y_f)) \wedge (z_1 \rightarrow (x_e \rightarrow \neg y_g)) \wedge (z_1 \rightarrow (x_f \rightarrow \neg y_a)) \wedge \\
&(z_1 \rightarrow (x_f \rightarrow \neg y_b)) \wedge (z_1 \rightarrow (x_f \rightarrow \neg y_g)) \wedge \\
&(z_2 \rightarrow (x_a \rightarrow \neg y_c)) \wedge (z_2 \rightarrow (x_a \rightarrow \neg y_d)) \wedge (z_2 \rightarrow (x_a \rightarrow \neg y_e)) \wedge \\
&(z_2 \rightarrow (x_b \rightarrow \neg y_c)) \wedge (z_2 \rightarrow (x_b \rightarrow \neg y_d)) \wedge (z_2 \rightarrow (x_b \rightarrow \neg y_e)) \wedge \\
&(z_2 \rightarrow (x_c \rightarrow \neg y_d)) \wedge (z_2 \rightarrow (x_c \rightarrow \neg y_e)) \wedge (z_2 \rightarrow (x_d \rightarrow \neg y_e)) \wedge \\
&(z_2 \rightarrow (x_e \rightarrow \neg y_d)) \wedge (z_2 \rightarrow (x_f \rightarrow \neg y_a)) \wedge (z_2 \rightarrow (x_f \rightarrow \neg y_c)) \wedge \\
&(z_2 \rightarrow (x_f \rightarrow \neg y_d)) \wedge (z_2 \rightarrow (x_f \rightarrow \neg y_e)) \wedge \\
&(z_3 \rightarrow (x_b \rightarrow \neg y_a)) \wedge (z_3 \rightarrow (x_b \rightarrow \neg y_g)) \wedge (z_3 \rightarrow (x_c \rightarrow \neg y_a)) \wedge \\
&(z_3 \rightarrow (x_c \rightarrow \neg y_b)) \wedge (z_3 \rightarrow (x_c \rightarrow \neg y_g)) \wedge \\
&(z_4 \rightarrow (x_e \rightarrow \neg y_b)) \wedge (z_4 \rightarrow (x_e \rightarrow \neg y_f)) \wedge (z_4 \rightarrow (x_f \rightarrow \neg y_b)) \wedge \\
&(z_4 \rightarrow (x_f \rightarrow \neg y_e)) \wedge \\
&(\neg q_1 \leftrightarrow \neg p_1 \vee y_d \vee \neg z_1) \wedge \\
&(\neg q_2 \leftrightarrow \neg p_2 \vee y_g \vee \neg z_2) \wedge \\
&(\neg q_3 \leftrightarrow \neg p_3 \vee y_d \vee y_e \vee y_f \vee \neg z_3) \wedge \\
&(\neg q_4 \leftrightarrow \neg p_4 \vee y_a \vee y_c \vee y_d \vee y_g \vee \neg z_4) \wedge \\
&(q_1 + q_2 + q_3 + q_4 \geq 2) \wedge \\
&(10 \times (q_1 + q_2 + q_3 + q_4) - 5 \times (p_1 + p_2 + p_3 + p_4) \geq 0)
\end{aligned}
$$

Its worth noticing that useful user constraints such as minimum (respectively maximum) conditions α and minimum (respectively maximum) consequences β in a $POSRule\ X \rightarrow Y$ could easily be modeled using the following formulas:

– *Maximum (respectively Minimum) Antecedent:* Formula (9) expresses that the number of items in the antecedent X of a *POSRule* shouldn't exceed (respectively has less than) α.

$$\sum_{a \in \Sigma} x_a \leq (resp. \geq) \quad \alpha \tag{9}$$

– *Maximum (respectively Minimum) Consequence:* Formula (10) expresses that the number of items in the consequence Y of a *POSRule* $X \rightarrow Y$ shouldn't exceed (respectively has less than) β.

$$\sum_{a \in \Sigma} y_a \leq (resp. \geq) \quad \beta \tag{10}$$

4 Two-Step Decomposition Approach for SAT Based Enumeration of POSRules

Extracting *POSRules* using the SAT-based approach described in Sect. 3 involves enumerating all models of the propositional formula $\Phi_{POSRules}(\mathcal{SD}, \lambda, \theta)$. The efficiency of this approach relies on how quickly these models can be enumerated. However, for large sequential databases $\mathcal{SD}$, the encoding can produce formulas with hundreds of thousands of variables and millions of clauses, which can significantly reduce both efficiency and scalability. In addition, the number of *POSRules* is typically very large, further slowing down the model enumeration process.

To address these challenges, we propose a decomposition-based approach to improve both scalability and efficiency of the proposed approach. This method splits the original dataset into smaller partitions $\mathcal{SD}_i$ and encodes each partition separately as $\Phi_{POSRules}(\mathcal{SD}_i, \lambda, \theta)$. Each partition produces a smaller formula with fewer items, which reduces computational complexity and allows faster model enumeration, improving the performance of the SAT-based approach introduced in Sect. 3.

We now describe the two-step decomposition approach that we propose for enumerating *POSRules* using SAT (see Algorithm 1). First, all items in the input sequential dataset $\mathcal{SD}$ are sorted in ascending order of support and stored in a vector $\mathcal{V}$. In the first decomposition step, we iterate over each item a in $\mathcal{V}$ and project $\mathcal{SD}$ into an intermediate partition $\mathcal{SD}'$ containing only sequences where a appears (iterations 25).

In the second step, for each item $a' \neq a$ appearing in $\mathcal{SD}'$, we project $\mathcal{SD}'$ into a final partition $\mathcal{SD}''$ containing sequences where both a and a' occur (iterations 78). Each final partition $\mathcal{SD}''$ is then encoded as $\Phi_{POSRules}(\mathcal{SD}'', \lambda, \theta)$, and all models of the resulting CNF formula are enumerated, with each model representing a *POSRule*. Variables x_a and $x_{a'}$ are set to true in the encoding of $\mathcal{SD}''$, while all items in $\mathcal{T}$ that have already been processed in previous iterations are set to false (iterations 812). This propagation prevents redundant

enumeration, ensuring that for each $\mathcal{SD}''$ we only enumerate rules of the form $X = aa'* \rightarrow Y$, where $*$ represents a subset of yet-unprocessed items appearing witnin $\mathcal{SD}''$.

At the end of the first step (iterations 1315), we handle the special case of *POSRules* of the form $a \rightarrow Y$ by encoding $\mathcal{SD}'$ with x_a set to true and all previously processed items set to false. All *POSRules* enumerated from different partitions are disjoint, and their union produces exactly the complete set of rules from the original dataset $\mathcal{SD}$. In the worst case, the number of generated encodings (i.e. partitions) is $O(|\Sigma|^2)$.

Finally, to improve efficiency, items with support less than λ are skipped, reducing the search space and accelerating the overall enumeration procedure. It is worth noting that the function *encode_cnf*() (lines 10 and 14) employs the encoding proposed in Sect. 3, whereas *enumerate_models*() (lines 11 and 15) relies on a slightly modified solver that adopts a DPLL-like procedure to enumerate all models of a given CNF formula [15].

Algorithm 1: Two-Step Decomposition-based *POSRules* Mining

Data: A sequence database $\mathcal{SD}$, Minimum support λ, Minimum confidence θ
Result: Set of all *POSRules* R

1 $R \leftarrow \emptyset$, $\Gamma \leftarrow \emptyset$, $V = \langle a_1 \dots a_n \rangle \leftarrow items(\mathcal{SD})$;
2 **for** $a \in V$ **do**
3 $\quad \mathcal{SD}' \leftarrow \{S_i \in \mathcal{SD} \mid a \in S_i\}$;
4 $\quad V' \leftarrow items(\mathcal{SD}') \setminus \{a\}$;
5 $\quad \Gamma \leftarrow \Gamma \cup \{a\}$;
6 $\quad \mathcal{T} \leftarrow \Gamma$
7 $\quad$ **for** $a' \in V'$ **do**
8 $\quad\quad \mathcal{SD}'' \leftarrow \{S_i \in \mathcal{SD}' \mid a' \in S_i\}$;
9 $\quad\quad \mathcal{T} \leftarrow \mathcal{T} \cup \{a'\}$;
10 $\quad\quad F = encode_cnf(\mathcal{T}, \mathcal{SD}'', \lambda, \theta)$
11 $\quad\quad R = R \cup enumerate_models(F)$;
12 $\quad\quad \mathcal{T} \leftarrow (\mathcal{T} \setminus \{a'\}) \cup \{\neg a'\}$;
13 $\quad$ **end**
14 $\quad F = encode_cnf(\mathcal{T}, \mathcal{SD}', \lambda, \theta)$
15 $\quad R = R \cup enumerate_models(F)$;
16 $\quad \Gamma \leftarrow (\Gamma \setminus \{a\}) \cup \{\neg a\}$;
17 **end**
18 **return** R;

5 Experimental Evaluation

To evaluate the proposed SAT-based approach for mining Partially Ordered Sequential Rules (*SPOSR*), we conducted a series of experiments on several real-world datasets. The performance of *SPOSR* was compared with that of state-of-the-art specialized algorithms, namely CMRule [10], CMDEO [9],RuleGrowth

[12], and ERMiner [11], all implemented in the SPMF tool[1]. The evaluation focuses on two criteria: the execution time required to extract all the desired *POSRules*, and the peak memory consumption during the mining process. We implemented *SPOSR* on top of the well-known SAT solver ManySAT [13], which is written in C++. The enumeration of all models (i.e., rules) is performed using a DPLL-based procedure [15]. All experiments were conducted on a machine equipped with an Intel i5 processor, 16 GB of RAM, and running at 2.30 GHz under Ubuntu 18.04. The *SPOSR* solver, along with the datasets used in this study, is available at the following link[2] to facilitate reproducibility.

Datasets

In these experiments, we used real-world datasets obtained from the SPMF website[3]. To ensure a comprehensive evaluation, we selected a diverse range of datasets, from small ones such as *SIGN* to large-scale datasets like *kosarak*, which contains approximately 1 million sequences and about 41 thousand distinct items. Table 3 summarizes the main characteristics of each dataset, including the number of sequences (#Sequences), the number of items (#Items), and the average sequence length (AGV.Length).

Table 3. Datasets characteristics

Dataset	#Sequences	#Items	AGV.Length
SIGN	730	310	51.9
retail best	4383	10157	48.23
retail all	4383	41431	32.15
Leviathan	5834	9494	33.8
E-shop	24026	317	61.9
MSNBC	31790	18	13.2
Bible	36369	14576	21.6
BMS1	59601	60978	2.1
BMS2	77512	330286	4.62
Kosarak	990002	41271	8.1

Experimental setup

To evaluate the behavior of *SPOSR* for different minimum support thresholds λ, we set a time limit of 1000 s and a memory limit of 15 GB. We then measure

[1] https://www.philippe-fournier-viger.com/spmf/index.php?link=algorithms.php.
[2] https://drive.google.com/drive/folders/1LtxhTnm93_skDtA0CbOzSyxHxYS1pyiO?usp=sharing.
[3] https://www.philippe-fournier-viger.com/spmf/index.php?link=datasets.php.

the performance (specifically, enumeration time and memory usage) of the algorithms (CMRule, CMDeo, RuleGrowth, ERMiner, and $SPOSR$) as a function of λ, while keeping the minimum confidence fixed at $\theta = 80\%$.

Results

Table 4. Enumeration time evaluation

Dataset	Min-supp	#Rules	Execution Time(s)				
			CMRule	CMDEo	RuleGrowth	ERMiner	SPOSR
SIGN	15%	174509	TO	TO	123.38	59.96	**5.62**
	10%	1083281	TO	TO	672.21	358.12	**18.51**
	5%	20986080	TO	TO	TO	TO	**260.26**
Leviathan	1%	43	310.54	894.89	89.12	95.75	**7.73**
	0.5%	292	TO	TO	295.27	410.87	**17.13**
	0.1%	733487	TO	TO	TO	TO	**156.17**
MSNBC	0.8%	1616	87.89	198.03	70.73	62.32	**2.60**
	0.5%	3440	131.01	667.61	206.62	153.02	**2.89**
	0.2%	5240	205.23	TO	527.00	355.82	**3.24**
Bible	0.5%	142	80.19	TO	136.00	120.98	**27.13**
	0.1%	27173	TO	TO	TO	MO	**138.95**
	0.05%	2042009	TO	TO	TO	MO	**279.11**
BMS1	0.08%	2744	9.20	17.22	4.56	3.66	**2.92**
	0.06%	1123969	TO	TO	131.24	46.29	**7.93**
	0.057%	20446658	TO	TO	TO	MO	**37.78**
BMS2	0.02%	868973	TO	TO	373.38	309.74	**48.13**
	0.01%	6310015	TO	TO	TO	TO	**104.55**
	0.005%	62931480	TO	TO	TO	TO	**268.11**
Kosarak	0.7%	126	41.43	208.89	71.54	**31.94**	34.55
	0.5%	286	80.30	568.70	135.36	**55.65**	57.72
	0.3%	1362	227.06	TO	359.96	**138.48**	142.42
retail best	2%	28770	TO	TO	/	101.82	**8.90**
	1%	677297	TO	TO	/	MO	**58.76**
	0.6%	7100174	TO	TO	/	MO	**385.25**
retail all	0.9%	28770	TO	TO	/	**103.14**	106.48
	0.7%	297448	TO	TO	/	377.48	**208.75**
	0.5%	2713178	TO	TO	/	MO	**469.16**
E-shop	20%	1433	TO	TO	/	170.97	**17.21**
	15%	9666	TO	TO	/	MO	**39.29**
	10%	142424	TO	TO	/	MO	**269.64**

Table 4 reports the results of the experiments, showing the execution times of the state-of-the-art algorithms CMRule, CMDEO, ERMiner, RuleGrowth, as well as

the proposed approach *SPOSR* for enumerating *POSRules*. As expected, the execution time increases as the minimum support decreases, which also corresponds to an increase in the number of *POSRules*. Overall, *SPOSR* outperforms the other algorithms across all datasets, except for *kosarak*, where *ERMiner* achieves slightly better results. Additionally, *CMDEO* and *CMRule* exceed the time limit (TO) in most cases, particularly for datasets containing sequences of itemsets, such as *E-shop*, *retail best* or *retail all*. It is also important to note that *RuleGrowth* was excluded from experiments on datasets with sequences of itemsets, as it does not handle such datasets correctly.

Figure 1 presents a scatter plot comparing *SPOSR* with other algorithms in terms of execution time across all experiments. It clearly shows that *SPOSR* consistently outperforms the competing approaches on every dataset. For a more detailed analysis, Figs. 2 and 3 illustrate the behavior of all algorithms with respect to both execution time and memory consumption. These figures depict, for each dataset, the evolution of enumeration time, memory usage, and the number of extracted rules as a function of the minimum support threshold.

Overall, the results indicate that, for the vast majority of datasets, *SPOSR* significantly outperforms the other algorithms in both runtime and memory usage, often by a large margin. For instance, on the e-shop dataset with a minimum support of 10%, *SPOSR* successfully enumerates all rules within 270 s, whereas the competing approaches exceed the timeout of 1000 s and ERMiner runs out of memory (MO).

In summary, *SPOSR* requires less time and memory to enumerate all *POSRules* compared to the other algorithms. This superior performance can be attributed to the decomposition technique (see Sect. 4), which accelerates the

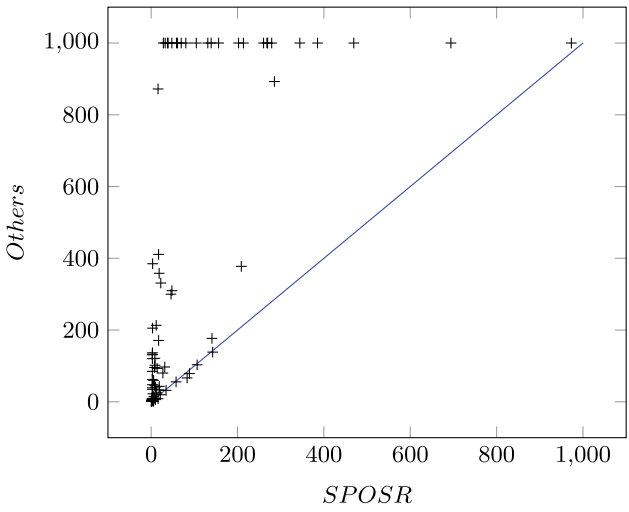

Fig. 1. Enumeration Time (in seconds): SPOSR vs Other Algorithms

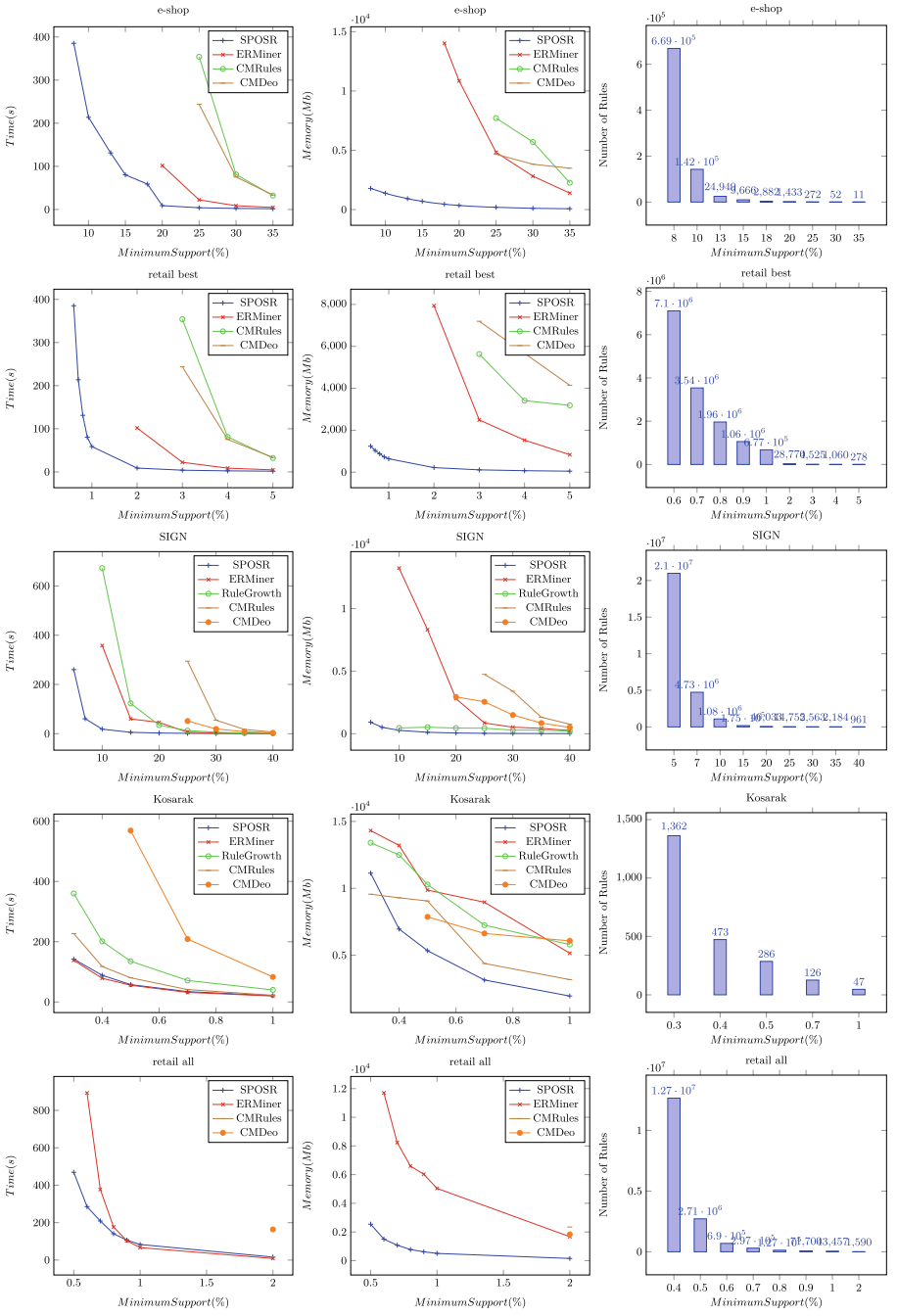

Fig. 2. Evaluation Results

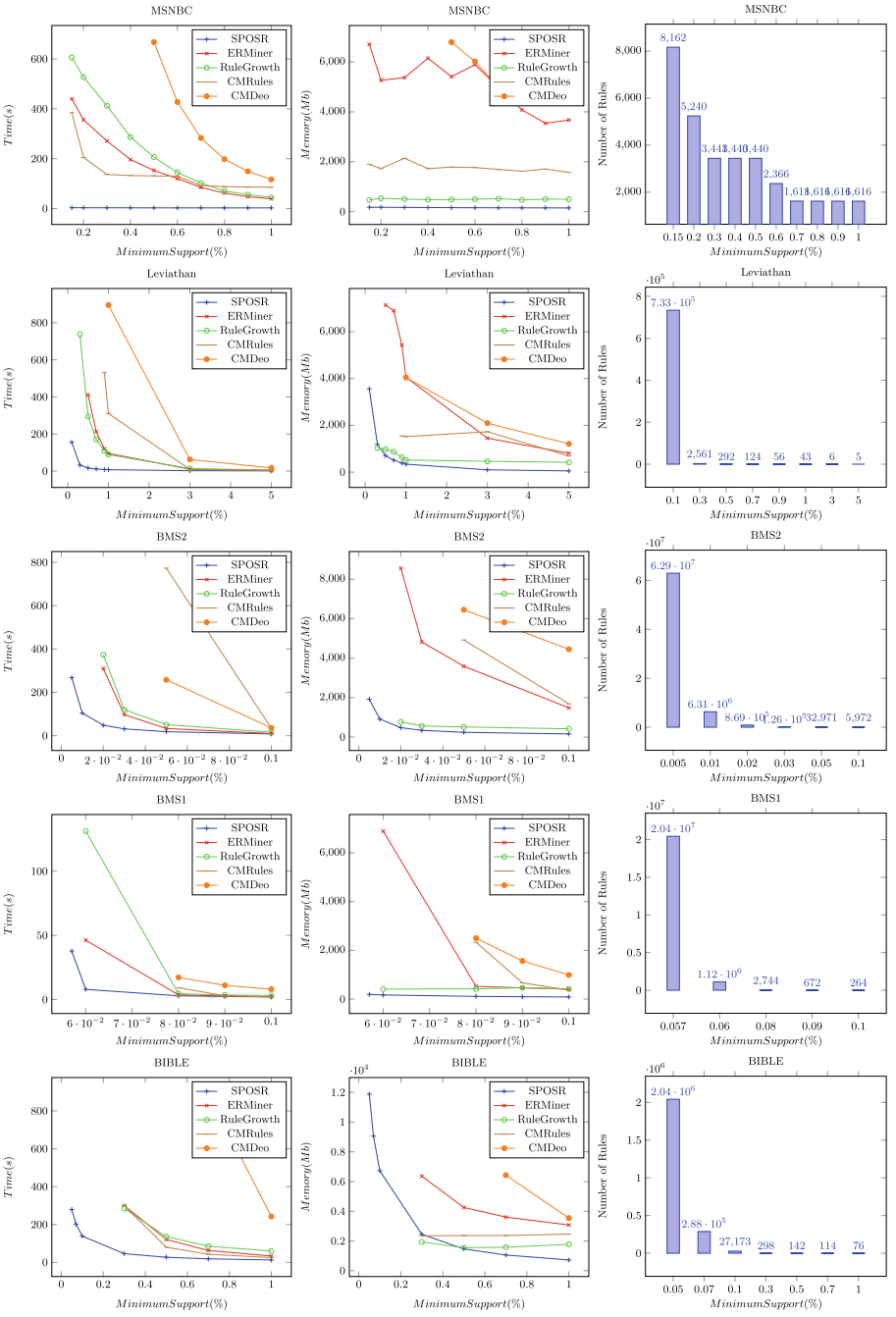

Fig. 3. Evaluation Results

enumeration process by pruning infrequent items from the search space, thereby reducing the size of the encoding.

6 Conclusion

In conclusion, this work addressed the problem of enumerating partially ordered sequential rules through a satisfiability-based framework. We introduced a SAT encoding that formalizes the enumeration of *POSRules* as a propositional logic formula, and demonstrated how a decomposition strategy can significantly enhance both the efficiency and scalability of the approach. Extensive experimental evaluations, along with a rigorous comparison against state-of-the-art methods, confirmed that the proposed approach consistently outperforms existing solutions in terms of execution time and memory usage across a wide range of datasets, including large-scale instances.

As a direction for future research, we aim to extend the proposed encoding to incorporate richer user-defined constraints and to enable the extraction of more compact and informative sets of sequential rules, such as minimal non-redundant rules.

References

1. Aoga, J.O.R., Guns, T., Schaus, P.: An efficient algorithm for mining frequent sequences with constraint programming. In: Frasconi, P., Landwehr, N., Manco, G., Vreeken, J. (eds.) Machine Learning and Knowledge Discovery in Databases (ECML PKDD 2016), Lecture Notes in Computer Science, pp. 315–330. Springer (2016)
2. Bailleux, O., Boufkhad, Y., Roussel, O.: New encodings of pseudo-Boolean constraints into cnf. In: Proceedings of the SAT, pp. 181–194 (2009)
3. Boudane, A., Jabbour, S., Sais, L., Salhi, Y.: A sat-based approach for mining association rules. In: Proceedings of the 25th International Joint Conference on Artificial Intelligence (IJCAI), pp. 2472–2478. IJCAI (2016)
4. Boudane, A., Ouzghara, R.: Compact sat-based encoding of mining minimal non-redundant association rules. In: Proceedings of the 2024 IEEE 18th International Conference on Application of Information and Communication Technologies (AICT), pp. 1–6. IEEE (2024)
5. Chen, Y., Fournier-Viger, P., Nouioua, F., Wu, Y.: Sequence prediction using partially-ordered episode rules. In: Proceedings of the IEEE ICDM Workshops, pp. 574–580. IEEE (2021)
6. Cook, S.A.: A short proof of the pigeonhole principle using extended resolution. SIGACT News **8**(4), 28–32 (1976)
7. De Raedt, L., Guns, T., Nijssen, S.: Constraint programming for itemset mining. In: Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD), pp. 204–212. ACM (2008)
8. Ezeife, C.I., Karlapalepu, H.: A survey of sequential pattern-based e-commerce recommendation systems. Algorithms **16**(10) (2023)

9. Fournier-Viger, P., Faghihi, U., Nkambou, R., Nguifo, E.M..: CMRULES: an efficient algorithm for mining sequential rules common to several sequences. In: Guesgen, H.W., Murray, R.C. (eds.) Proceedings of the 23rd International Florida Artificial Intelligence Research Society Conference (FLAIRS), pp. 106–111. AAAI Press (2010)
10. Fournier-Viger, P., Faghihi, U., Nkambou, R., Nguifo,E.M.: CMRules: mining sequential rules common to several sequences. Knowl. Based Syst. **25**(1), 63–76 (2012)
11. Fournier-Viger, P., Gueniche, T., Zida, S., Tseng, V.S.: ERMiner: Sequential Rule Mining Using Equivalence Classes. In: Blockeel, H., van Leeuwen, M., Vinciotti, V. (eds.) IDA 2014. LNCS, vol. 8819, pp. 108–119. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-12571-8_10
12. Fournier-Viger, P., Cheng-Wei, W., Tseng, V.S., Cao, L., Nkambou, R.: Mining partially-ordered sequential rules common to multiple sequences. IEEE Trans. Knowl. Data Eng. **27**(8), 2203–2216 (2015)
13. Hamadi, Y., Jabbour, S., Sais, L.: ManySAT: a parallel sat solver. J. Satisfiability Boolean Model. Comput. **6**(4), 245–262 (2009)
14. Izza, Y., Jabbour, S., Raddaoui, B., Boudane, A..: On the enumeration of association rules: a decomposition-based approach. In: Proceedings of the 29th International Joint Conference on Artificial Intelligence (IJCAI), pp. 1265–1271. IJCAI (2020)
15. Jabbour, S., Lonlac, J., Sais, L., Salhi, Y..: Extending modern sat solvers for model enumeration. In: Proceedings of the IEEE International Conference on Information Reuse and Integration (IRI), pp. 803–810. IEEE (2014)
16. Jabbour, S., Mhadhbi, N., Raddaoui, B., Sais, L.: Sat-based models for overlapping community detection in networks. Computing **102**(5), 1275–1299 (2020)
17. Lee, J.M., Hauskrecht, M.: Neural clinical event sequence prediction through personalized online adaptive learning. In: Tucker, A., et al., (eds.) Artificial Intelligence in Medicine, pp. 175–186. Springer (2021)
18. Sinz, C.: Towards an optimal CNF encoding of Boolean cardinality constraints. In: Proceedings of the International Conference on Principles and Practice of Constraint Programming, pp. 827–831 (2005)
19. Tseitin, G.S.: On the complexity of derivations in the propositional calculus. Stud. Math. Math. Logic, 115–125 (1968)
20. Zaki, M.J.: SPADE: an efficient algorithm for mining frequent sequences. Mach. Learn. **42**(1–2), 31–60 (2001)