Information Systems 34 (2009) 438–453

Contents lists available at ScienceDirect

Information Systems

journal homepage: www.elsevier.com/locate/infosys

Non-redundant sequential rules—Theory and algorithm

David Lo [a,*], Siau-Cheng Khoo [b], Limsoon Wong [b]

[a] *School of Information Systems, Singapore Management University, Singapore*
[b] *Department of Computer Science, National University of Singapore, Singapore*

A R T I C L E I N F O

Article history:
Received 8 January 2009
Accepted 21 January 2009
Recommended by: D. Shasha

Keywords:
Theoretical data mining
Frequent pattern mining
Sequential pattern mining
Sequential rules
Non-redundant rules

A B S T R A C T

A sequential rule expresses a relationship between two series of events happening one after another. Sequential rules are potentially useful for analyzing data in sequential format, ranging from purchase histories, network logs and program execution traces.

In this work, we investigate and propose a syntactic characterization of a non-redundant set of sequential rules built upon past work on compact set of representative patterns. A rule is redundant if it can be inferred from another rule having the same support and confidence. When using the set of mined rules as a composite filter, replacing a full set of rules with a non-redundant subset of the rules does not impact the accuracy of the filter.

We consider several rule sets based on composition of various types of pattern sets—generators, projected-database generators, closed patterns and projected-database closed patterns. We investigate the completeness and tightness of these rule sets. We characterize a tight and complete set of non-redundant rules by defining it based on the composition of two pattern sets. Furthermore, we propose a compressed set of non-redundant rules in a spirit similar to how closed patterns serve as a compressed representation of a full set of patterns. Lastly, we propose an algorithm to mine this compressed set of non-redundant rules. A performance study shows that the proposed algorithm significantly improves both the runtime and compactness of mined rules over mining a full set of sequential rules.

1. Introduction

Sequential pattern mining first proposed by Agrawal and Srikant [1] has been the subject of active research [2–7]. Given a database containing sequences, sequential pattern mining identifies sequential patterns appearing with enough support. It has potential application in many areas such as analysis of market data, purchase histories, web logs, etc.

Sequential rules express temporal relationships among patterns [8]. It can be considered as a natural extension to sequential patterns, as association rules are to frequent itemsets [9]. A sequential rule expressed as *pre* $\rightarrow$ *post*, specifies that there is sufficiently high confidence that the pattern *post* will occur in sequences following an occurrence of *pre*. Compared to sequential patterns, rules allow better understanding of temporal behaviors exhibited in a sequence database. Consider a classic example of purchasing behavior in a video shop [1]: a customer who buys Star Wars episode IV will likely buy episodes V and VI in the future. The purchase pattern $\langle IV, V, VI\rangle$ is the pattern showing the purchase behavior. However, imagine a standard video shop with hundreds of buyers with various preferences. The pattern $\langle IV, V, VI\rangle$ will tend to occur with a low support. Mining with low support will return the pattern, however, typically along with many irrelevant or spurious patterns. Rules can throw away

* Corresponding author. Tel.: +65 98421014.
E-mail addresses: davidlo@smu.edu.sg (D. Lo), khoosc@comp.nus.edu.sg (S.-C. Khoo), wongls@comp.nus.edu.sg (L. Wong).

0306-4379/$ - see front matter © 2009 Elsevier B.V. All rights reserved.
doi:10.1016/j.is.2009.01.002

many spurious patterns by introducing the notion of confidence to the set of patterns. Only rules satisfying both support and confidence thresholds are mined.

Sequential rules extend the usability of patterns beyond the understanding of sequential data. A mined rule represents the *constraint* that its premise is followed by its consequent in sequences. Hence, rules are potentially useful for detecting and filtering anomalies which violate the corresponding constraints. They have applications in detecting errors, intrusions, bugs, etc. Mining rule-like sequencing constraints from sequential data has been shown useful in medicine (e.g., [10]) and software engineering (e.g., [11–13]) domains. Some examples of useful rules include:

1. (Market data) If a customer buys a car, he/she will eventually buy car insurance. This is potentially useful in designing personalized marketing strategy.
2. (Medical data) If a patient has a fever, which is followed by a drop in thrombosite level and followed by appearance of red spots in the skin, then it is likely that the patient will need a treatment for dengue fever. This is potentially useful in predicting a suitable type of treatment needed for a patient.
3. (Software data) If a Windows device driver calls *KeAcquireSpinLock*, then it eventually needs to call *KeReleaseSpinLock* [14].

Spiliopoulou [8] proposes generating a *full set* of sequential rules (i.e., all frequent and confident rules) from a *full set* of sequential patterns (i.e., all frequent patterns). Generating a full set of sequential rules can be very expensive. The number of frequent patterns is exponential to the maximum pattern length: if a sequential pattern of length l is frequent, all its $\mathcal{O}(2^l)$ subsequences are frequent as well. Each frequent pattern of length l can possibly generate $l-1$ rules (depending on the minimum confidence threshold). Hence, there is an exponential growth in the number of rules with respect to the maximum pattern length.

To tame the explosive growth of rules, we propose mining a *non-redundant* set of sequential rules. Central to our method is the notion of *rule inference*. This notion is used to define and remove redundancy among rules. When using the set of mined rules as a composite filter, replacing a full set of rules with the non-redundant subset of rules does not impact the accuracy of the filter.

There have been many studies on mining frequent sequential patterns [1–5,15–17]. These studies include those mining a compact representation of patterns, referred to as closed patterns [6,7] and generators [18,19]. These compact representative patterns can be mined with much more efficiency than the full set of frequent patterns. However, there *has not been* any study relating these compact representative patterns with a non-redundant set of sequential rules. In particular the following questions need to be addressed: Can a non-redundant set of rules be obtained from compact representative patterns? What types of compact representative patterns need to be mined to form non-redundant rules? What do we mean by a non-redundant set of rules? Can we characterize the non-redundant set of rules? How to use representative patterns to form non-redundant rules? How much effort is needed to obtain a non-redundant set of rules from compact representative patterns? Can we design an efficient algorithm to obtain a non-redundant set of rules from patterns?

In this paper, we address the above research questions. We focus on performing an investigation and a characterization of a set of non-redundant sequential rules built upon existing studies on compact sets of representative sequential patterns. In addition, we propose an algorithm, develop a tool, and perform a performance study on mining a non-redundant set of sequential rules.

We investigate four different sets of patterns, namely generators, projected-database generators, closed patterns and projected-database closed patterns. For the projected-database generators and closed patterns, aside from the format and support values of patterns, we also consider their projected database (cf., [3,6]).

A rule set can be formed by composing patterns. We investigate various configurations of compositions of the above four sets of patterns. These sets are then evaluated based on the two criteria of completeness and tightness. A rule set is complete, if each frequent and confident rule can be inferred by one of the rules in the rule set. A rule set is tight, if the set contains no redundant rules. We characterize a tight and complete set of non-redundant rules based on these configurations.

Additionally, to further reduce the number of mined rules, we propose a rule compression strategy to compress the set of non-redundant rules. This strategy is in the same spirit as how closed patterns are used as a compressed representation of a full set of frequent patterns.

We propose an algorithm to mine this compressed set of non-redundant rules. Our performance study shows much benefit in mining non-redundant rules over a full set of rules. The study shows that the runtime and number of rules mined can be reduced by up to 5598 *times* and 8583 *times*, respectively!

The contributions of our work are as follows:

1. We propose a concept of non-redundant rules based on logical inference.
2. We investigate different sets of patterns and their various compositions to form different sets of rules. We study the quality of these rule sets with respect to completeness and tightness.
3. We characterize a tight and complete set of non-redundant rules based on compositions of patterns.
4. We propose and characterize compression of the non-redundant set of rules.
5. We develop an algorithm to mine the compressed set of non-redundant rules and show that it performs much faster than mining a full set of sequential rules.

The outline of this paper is as follows. Section 2 presents terminologies and definitions used. Among other things this section defines the meaning of closed pattern, generator, projected database, equivalence class, rule satisfiability and support and confidence values of rules. Section 3 describes some important properties of pattern-sets and also of rule inference. These properties are needed in later sections to show that a set of rules is a complete and tight set of non-redundant rules. Section 4 describes various configuration of rules by composing various pattern sets and characterizes them with respect to completeness and tightness. This section also characterizes a tight and non-redundant set of rules by composition of two different pattern-sets. Section 5 describes the concept of compressed set of rules. Section 6 describes our algorithm to mine a compressed set of non-redundant rules. Section 7 describes our performance study. Section 8 compares and contrasts our method and contribution with related works. Section 9 discusses issues on uniqueness of a tight and complete set of non-redundant rules and a more complex rule inference strategy. Section 10 concludes this paper.

2. Definitions

Let I be a set of distinct items. Let a *sequence* S be an ordered list of events. We denote S by $\langle \mathtt{e}_1, \mathtt{e}_2, \ldots, \mathtt{e}_{\mathtt{end}} \rangle$ where each $\mathtt{e}_\mathtt{i}$ is an item from I. A pattern $P_1 = \langle \mathtt{e}_1, \mathtt{e}_2, \ldots, \mathtt{e}_\mathtt{n} \rangle$ is considered a *subsequence* of another pattern P_2 ($\langle \mathtt{f}_1, \mathtt{f}_2, \ldots, \mathtt{f}_\mathtt{m} \rangle$), denoted by $P_1 \sqsubseteq P_2$, if there exist integers $1 \leqslant \mathtt{i}_1 < \mathtt{i}_2 < \mathtt{i}_3 < \mathtt{i}_4 < \cdots < \mathtt{i}_\mathtt{n} \leqslant \mathtt{m}$ such that $\mathtt{e}_1 = \mathtt{f}_{\mathtt{i}_1}, \mathtt{e}_2 = \mathtt{f}_{\mathtt{i}_2}, \ldots, \mathtt{e}_\mathtt{n} = \mathtt{f}_{\mathtt{i}_\mathtt{n}}$. We also say that P_2 is a *super-sequence* of P_1.The sequence database under consideration is denoted by *SeqDB*. The length of P is denoted by $|P|$. A pattern P_1 ++P_2 denotes the concatenation of pattern P_1 and pattern P_2.

The *absolute support* of a pattern w.r.t. to a sequence database *SeqDB* is the number of sequences in *SeqDB* that are super-sequences of the pattern. The *relative support* of a pattern w.r.t. to *SeqDB* is the ratio of its absolute support to the total number of sequences in *SeqDB*. The support (either absolute or relative) of a pattern P is denoted by $sup(P, SeqDB)$. We ignore the mentioning of the database when it is clear from the context.

Definition 2.1 (*Projected database*). A sequence database *SeqDB projected* on a pattern P is defined as: $SeqDB_P = \{s_x | S \in SeqDB, S = p_x{+}{+}s_x, \quad p_x$ is the minimum prefix of S containing $P\}$ (cf., [3]).

To illustrate the concept of projected database, consider the example database *ExDB* shown in Table 1. As examples, projected databases w.r.t. *ExDB* on patterns $\langle A, B \rangle$ and $\langle A, B, C \rangle$ are shown in Tables 2 and 3, respectively.

Table 1
Example database—ExDB.

ID	Sequence
S1	$\langle A, B, C, B, D, A, B, C, D \rangle$
S2	$\langle A, B, E, C, F, D \rangle$
S3	$\langle A, B, F, C, D, E \rangle$

Table 2
$ExDB_{\langle A,B \rangle}$.

$\langle C, B, D, A, B, C, D \rangle$
$\langle E, C, F, D \rangle$
$\langle F, C, D, E \rangle$

Table 3
$ExDB_{\langle A,B,C \rangle}$.

$\langle B, D, A, B, C, D \rangle$
$\langle F, D \rangle$
$\langle D, E \rangle$

Projected database provides the series of events occurring after pattern instances. It provides the *context* where a pattern can be extended or grown further. Two patterns p and q having the same projected database, have the same context, ensuring that any event happening after an instance p will also appear after the corresponding instance of q in the database.

Definition 2.2 (*Frequent, CS-, LS-Closed*). A pattern P is considered *frequent* in SeqDB when its support, $sup(P, SeqDB)$, exceeds a minimum threshold *min_sup*. A frequent pattern P is considered to be *closed* if there exists no proper super-sequence of P having the same support as P [6,7]. A frequent pattern P is considered to be *projected-database* closed if there exists no proper super-sequence of P having the same support and projected database as P [6]. To avoid ambiguity we call the set of closed patterns CS-Closed and the set of projected-database-closed patterns LS-Closed. Note that CS-Closed $\subseteq$ LS-Closed.

Definition 2.3 (*CS-, LS-Key*). A frequent pattern P is considered to be a *generator* in *SeqDB* if there exists no proper sub-sequence of P having the same support as P in *SeqDB* [18,19]. A frequent pattern P is considered to be a *projected-database* generator if there exists no proper sub-sequence of P having the same support and projected database as P. To avoid ambiguity we call the set of generators CS-Key and the set of projected-database generators LS-Key. Note that CS-Key $\subseteq$ LS-Key.

Projected-database-closed pattern (LS-Closed) was first introduced in [6]. Both LS-Closed and LS-Key are interesting concepts, as subsumed patterns (i.e., non-LS-Closed or non-LS-Key patterns) and the corresponding representative pattern (i.e., corresponding LS-Closed or LS-Key pattern) have the same context, ensuring that the series of events appearing after corresponding pattern instances to be the same.

Let us also define two different concepts of equivalence classes of sequential patterns as follows:

Definition 2.4 (*EQClass (P,CS,SeqDB)*). Two patterns P_X and P_Y are in the same *equivalence class* w.r.t. *SeqDB* iff, for all s in *SeqDB*, we have $P_X \sqsubseteq s$ iff $P_Y \sqsubseteq s$.

We denote the equivalence class of P_X in *SeqDB* by $EQClass(P_X, CS, SeqDB)$ and $EQClass(P_X, CS)$ when there is no ambiguity.

Definition 2.5 (*EQClass (P,LS,SeqDB)*). Two patterns P_X and P_Y are in the same *projected-database equivalence class* w.r.t. *SeqDB* iff

1. for all s in *SeqDB*, we have $P_X \sqsubseteq s$ iff $P_Y \sqsubseteq s$; and
2. P_X and P_Y have the same projected database in *SeqDB*.

We denote the projected-database equivalence class of P_X in *SeqDB* by *EQClass* $(P_X, LS, SeqDB)$ and *EQClass* (P_X, LS) when there is no ambiguity.

Example. To illustrate the concepts of equivalence class, generator (CS-Key), projected-database generator (LS-Key), closed pattern (CS-Closed) and projected-database closed pattern (LS-Closed) consider the database shown in the following table.

Seq ID.	Sequence	Seq ID.	Sequence
*S*1	⟨A, D, A⟩	*S*2	⟨B, A, D, A⟩
*S*3	⟨A, B, C, B⟩	*S*4	⟨A, B, B, C⟩
*S*5	⟨B, B, A, B⟩	*S*6	⟨D, X, Y⟩

Considering *min_sup* set at 2, the frequent pattern space corresponds to the following lattice in Fig. 1. There are 16 frequent patterns including the empty pattern ⟨ ⟩ which is trivially frequent; we ignore this trivial pattern in subsequent discussion.

Among the 16 frequent patterns, there are eight equivalence classes (i.e., EQClass(·,CS,·)) marked by the dashed lines and referred to as EQ_1–EQ_8 in Fig. 1. For projected-database equivalence classes, since we need to check for equivalence of projected database as well, we need to split EQ_5 and EQ_8, each into two projected-database equivalence classes (i.e., EQClass(·,LS,·)). The newly introduced projected-database equivalence classes EQ_{5-2} and EQ_{8-2} are shown by solid red circle. Let us refer to the other corresponding projected-databases as EQ_{5-1} and EQ_{8-1}, respectively. Other equivalence classes EQ_1, EQ_2, EQ_3, EQ_4, EQ_6 and EQ_7 are also projected-database equivalence classes.

In each equivalence class, generally those patterns at the bottom are the closed patterns while those at the top are the generators. For example, consider the equivalence class EQ_5 which is supported by *S*1 and *S*2 in the database. The closed pattern of EQ_5 is $\langle A,D,A\rangle$, while the set of generators is $\{\langle A,A\rangle, \langle A,D\rangle, \langle D,A\rangle\}$. Also consider EQ_8 which is supported by *S*3 and *S*4 in the database. The set of closed patterns of EQ_8 is $\{\langle A,B,B\rangle, \langle A,B,C\rangle\}$. The set of generators of EQ_8 is the set $\{\langle A,B,B\rangle, \langle C\rangle\}$.

The case is similar with projected-database equivalence class. For example, consider the equivalence class EQ_{5-1}. The closed pattern of EQ_{5-1} is $\langle A,D,A\rangle$, while the set of generators is $\{\langle A,A\rangle, \langle D,A\rangle\}$. Also consider EQ_{8-1}. The closed patterns of EQ_{8-1} is $\langle A,B,C\rangle$. The generator of EQ_8 is $\langle C\rangle$.

Note that an equivalence class can have more than one closed pattern and more than one generator. Similarly, it can be easily seen that a projected-database equivalence class can have more than one projected-database closed pattern and more than one projected-database generator.

Similar to Wang and Han [7], we consider only *single-item* sequences. This simplifies our presentation. Furthermore, single-item sequences also represent many important types of sequences such as web click streams, purchase histories, program API traces, etc.

From the definitions of "support" and "frequent pattern", sequential patterns possess "apriori" property [20]: if a sequential pattern is frequent then all its subsequences are also frequent. In other words, support of a pattern is greater or equal to support of its super-sequences.

Definition 2.6 (*Pattern matching*). A sequence *S* is said to *match* a pattern *P* iff $P \sqsubseteq S$. This is denoted by $P\langle\langle S\rangle\rangle$. The inverse, that *S* does not match *P*, is denoted by $\neg P\langle\langle S\rangle\rangle$.

As an example, consider a pattern $P = \langle A,B\rangle$ and two sequences $S1 = \langle A,C,B\rangle$ and $S2 = \langle C,D\rangle$. The first sequence *S*1 *matches* *P* since $P \sqsubseteq S1$. The second sequence *S*2 *does not match* *P* since $P \not\sqsubseteq S2$.

Rules are different from patterns. Rules are composed of two parts: pre and post-conditions. A rule asserts that *if* a series of events occurs, *then* another series of events must occur later in the sequence. Formally,

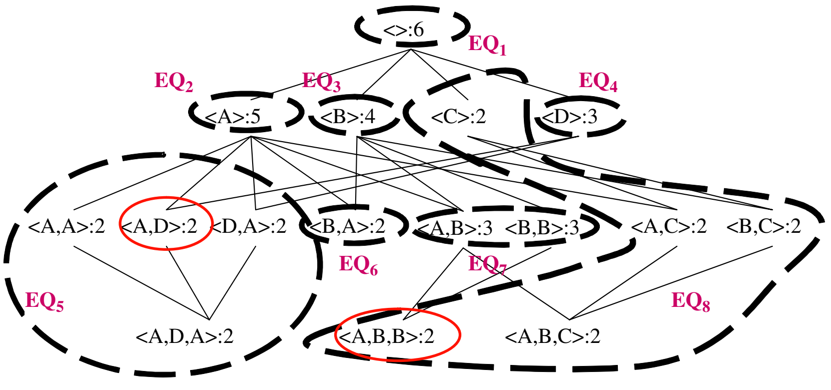

Fig. 1. Frequent pattern space and equivalence classes.

Definition 2.7 (*Rule satisfiability*). A sequence S is said to *satisfy* a sequential rule r of the form $pre \rightarrow post$ if either one of the following two cases holds:

1. It matches the pre-condition and subsequently the post-condition of the rule, i.e., $\exists S_1, S_2.\ S = S_1{++}S_2 \wedge pre\langle\langle S_1\rangle\rangle \wedge post\langle\langle S_2\rangle\rangle$.
2. It does not match the pre-condition of the rule, i.e., $\nexists S_1, S_2.\ S = S_1{++}S_2 \wedge pre\langle\langle S_1\rangle\rangle$.

A sequence S satisfying a rule r is denoted by $r\langle\langle S\rangle\rangle$; otherwise, it is denoted by $\neg r\langle\langle S\rangle\rangle$.

Following from the pattern example above, consider a rule $r = A \rightarrow B$ and two sequences $S1 = \langle \mathtt{A}, \mathtt{C}, \mathtt{B}\rangle$ and $S2 = \langle \mathtt{C}, \mathtt{D}\rangle$. $S1$ satisfies r since B occurs after the occurrence of A in $S1$. In contrast to the pattern example, $S2$ satisfies r since we *cannot* find any A in $S2$.

Forming *sequential rules* from frequent sequential patterns under the support-confidence framework is analogous to forming association rules from frequent itemsets. A sequential rule is denoted by $r = X \rightarrow Y(s, c)$, where X and Y are sequential patterns and s and c are the support and confidence values [8,20]. We omit the support and confidence values if it is clear or irrelevant to the context. A rule $r = X \rightarrow Y$ is constructed from two sequential patterns: X and $X{++}Y$. The *confidence* of r, denoted by $conf(r)$, is defined as the ratio of $sup(X{++}Y)$ to $sup(X)$. On the other hand, the *support* of r, denoted by $sup(r)$, is defined to be equal to $sup(X{++}Y)$. Formally, we define support and confidence in Definition 2.8.

Definition 2.8 (*Support and confidence*). A rule r_X has support equal to the number of sequences in *SeqDB* that matches the pre-condition, and subsequently the post-condition of the rule. Also, its confidence is equal to the likelihood of sequences matching the pre-condition of r_X to also subsequently match the post-condition of r_X.

Note that from the above definition, for a rule r_X, it can be seen that only two sets of sequences in the input sequence database *SeqDB* affect the significance values of r_X. The first set is sequences in *SeqDB* that satisfies r_X by matching the pre-condition and subsequently the post-condition of r_X. The second set is sequences in *SeqDB* that violates r_X by matching the pre-condition but not subsequently the post-condition.

A sequential rule consists of four components: an identifier, a description, a support value and a confidence value. This is denoted by "identifier = description (support, confidence)", e.g., $R = a \rightarrow b(0.2, 0.8)$. Some of these components may be omitted if they are irrelevant or clear from the context. Given a rule $r = X \rightarrow Y(s, c)$, we denote the pre-/post-condition of r (i.e., X/Y) by $r.Pre/r.Post$.

Formally, we also define significant rules, i.e., frequent and confident rules in Definition 2.9.

Definition 2.9 (*Rule significance*). A rule with support higher than a threshold *min_sup* is considered *frequent*. A rule with confidence higher than a threshold *min_conf* is considered *confident*.

3. Inference, redundancy and pattern properties

Our approach to mining a non-redundant set of rules lies in a construction based on *rule inference*. In this section we define rule inference and mention properties relating to pattern-sets and rule inference.

3.1. Definitions of inference and redundancy

Definition 3.1 (*Rule inference*). Given a sequence database *SeqDB* and two rules r_1, r_2. r_1 is said to *infer* r_2 if and only if:

1. $r_2\langle\langle S\rangle\rangle$ whenever $r_1\langle\langle S\rangle\rangle$, for every sequence S regardless of S's presence in *SeqDB*.
2. $sup(r_1, SeqDB) = sup(r_2, SeqDB)$ and $conf(r_1, SeqDB) = conf(r_2, SeqDB)$.

Definition 3.2 (*Redundant rules*). A rule is said to be *redundant* in a set of rules R iff it can be inferred by another rule in R.

Consider the following two rules: $r_1 = \langle \mathtt{A}\rangle \rightarrow \langle \mathtt{B}, \mathtt{C}, \mathtt{D}\rangle$ and $r_2 = \langle \mathtt{A}\rangle \rightarrow \langle \mathtt{B}\rangle$ having the same support and confidence. r_2 is redundant since it can be *inferred* by r_1.

3.2. Properties of pattern sets and inference

We now identify some properties associated with patterns. We then leverage on these properties to highlight the properties of rule inference and rule coverage.

Property 1. *Let P_X be a pattern of a sequence database SeqDB. If P_X is frequent, then: $sup(P_X) = Max_{CP \in CS\text{-}Closed \wedge P_X \sqsubseteq CP}.\ (sup(CP))$.*

Property 2. *Consider two patterns P_X and P_Y appearing in a sequence database SeqDB. If $P_X \sqsubseteq P_Y$ and $sup(P_X) = sup(P_Y)$, then P_X and P_Y are supported by the same set of sequences.*

Property 3 (*Transitivity*). *If r_X infers r_Y and r_Y infers r_Z then r_X infers r_Z.*

Before stating the necessary and sufficient condition of rule inference, we state two lemmas.

Lemma 1. *Suppose for every sequence s, we have $s \sqsupseteq b$ when $s \sqsupseteq a$, then it must be the case that $a \sqsupseteq b$.*

Lemma 2. *Suppose for every sequence s, we have $s \not\sqsupseteq b$ when $s \not\sqsupseteq a$, then it must be the case that $a \sqsubseteq b$.*

The sufficient and necessary condition of rule inference is as follows.

Property 4 (*Sufficient and necessary inference*). *Given $r_X = pre_X \rightarrow post_X$ and $r_Y = pre_Y \rightarrow post_Y$ generated from SeqDB. r_Y infers r_X if and only if the following four conditions hold, (1) $pre_Y \sqsubseteq pre_X$ (2) $pre_Y{++}post_Y \sqsupseteq pre_X{++}post_X$ (3) $sup(r_Y) = sup(r_X)$ and (4) $conf(r_Y) = conf(r_X)$.*

Proof. *The right-to-left direction*: Suppose the four conditions holds. Condition (1) ensures that whenever pre_Y

does not hold, pre_X will also not hold. This implies that whenever r_Y holds (vacuously), r_X also holds (vacuously). Condition (2) ensures that whenever $pre_Y{+}{+}post_Y$ holds, $pre_X{+}{+}post_X$ also holds. This implies whenever r_Y holds (not vacuously), r_X also holds. Conditions (3) and (4) ensure that r_Y has the same support and confidence as r_X. Hence, the above are sufficient condition for inference (cf., Definition 3.1).

The left-to-right direction: Suppose r_Y *infers* r_X. Then r_Y and r_X have equal support and confidence. Hence, conditions (3) and (4) hold. We only need to prove that conditions (1) and (2) hold. For any sequence s, we have $\neg r_Y\langle\langle s\rangle\rangle$ iff $pre_Y \sqsubseteq s$ and $pre_Y{+}{+}post_Y \not\sqsubseteq s$, and $\neg r_X\langle\langle s\rangle\rangle$ iff $pre_X \sqsubseteq s$ and $pre_X{+}{+}post_X \not\sqsubseteq s$. Taking the contrapositive of r_Y infers r_X, we have $\neg r_Y\langle\langle s\rangle\rangle$ whenever $\neg r_X\langle\langle s\rangle\rangle$. Thus $pre_Y \sqsubseteq s$ whenever $pre_X \sqsubseteq s$, and $pre_Y{+}{+}post_Y \not\sqsubseteq s$ whenever $pre_X{+}{+}post_X \not\sqsubseteq s$. As s is arbitrary, by Lemma 1, we conclude $pre_Y \sqsubseteq pre_X$, proving condition 1. Also, by Lemma 2, we conclude $pre_X{+}{+}post_X \sqsubseteq pre_Y{+}{+}post_Y$, proving condition (2). □

As an example, let *SeqDB* be a database of two sequences: $\langle \mathrm{A,B}\rangle$ and $\langle \mathrm{A,B,D}\rangle$. Let r_1 be $\langle \mathrm{A}\rangle \rightarrow \langle \mathrm{B}\rangle$, r_2 be $\langle \mathrm{A}\rangle \rightarrow \langle \mathrm{B,D}\rangle$ and r_3 be $\langle \mathrm{A,B}\rangle \rightarrow \langle \mathrm{D}\rangle$. Then r_2 and r_3 satisfy the sufficient inference property. Hence, r_2 infers r_3. However, r_1 does not infer r_2 and r_2 does not infer r_1, as r_1 and r_2 have different support and confidence.

4. Theory of non-redundant rules

Generating sequential rules from a full set of sequential patterns can be exorbitant. Yan et al. [6] have shown that the size of a full set of sequential patterns is exponential to the maximum length of patterns in the closed pattern set.

We therefore advocate using *generator* and *closed pattern sets* (either regular or projected-database) to generate sequential rules. Furthermore, instead of generating all frequent and confident sequential rules, we strive to generate a *non-redundant set of sequential rules* from which all other rules can be derived. There are three technical challenges pertaining to this research direction: (1) the assurance that all interesting rules can be logically inferred from this non-redundant set (i.e., the non-redundant set is complete), (2) the assurance that the set of non-redundant rules is tight, and (3) the need to compute the support and confidence of all rules efficiently.

The following sub-sections analyze various configurations of rule sets in terms of completeness and tightness by composing different pattern sets. A configuration that results in a tight and complete set of non-redundant rules is then identified.

4.1. Characterization of non-redundant rules

Based on the four pattern sets defined in Section 2—LS-Key, LS-Closed, CS-Key and CS-Closed—we can form different sets of rules by composing these patterns to infer all other frequent and confident rules. The purpose of this section is to characterize these rules with respect to two properties namely:

1. Completeness: All frequent and confident rules can be inferred from the generated set of rules.
2. Tightness: There exists no two different rules r and r' in the final set of rules where r infers r'.

Definition 4.1 (*Configuration* (X, Y)). Based on the above four pattern sets, we define Configuration (X, Y), where both X and Y can be any of the four pattern sets above, as the following set of rules: $\{pre \rightarrow post | pre \in X$, there is $sh{+}{+}post \in Y$, $sup(sh{+}{+}post) = sup(pre{+}{+}post)$, $pre \sqsubseteq sh$ and sh is as short as possible$\}$.

Each of the above configurations forms a set of rules. There are in total 16 different configurations. Abstracting away LS- and CS-, we have four meta configurations: (Key, Key), (Key, Closed), (Closed, Key) and (Closed, Closed). Each meta configuration corresponds to four configurations. We investigate only (Key, Closed) and (Closed, Closed) as they shed light on the non-redundant set of rules and the compressed set of non-redundant rules. In particular we investigate the following configurations:

1. Configuration (LS-Key, LS-Closed)
2. Configuration (LS-Key, CS-Closed)
3. Configuration (CS-Key, LS-Closed)
4. Configuration (CS-Key, CS-Closed)
5. Configuration (LS-Closed, LS-Closed)
6. Configuration (LS-Closed, CS-Closed)
7. Configuration (CS-Closed, LS-Closed)
8. Configuration (CS-Closed, CS-Closed)

Lemma 3. *Configuration (CS-Key, CS-Closed) is not tight.*

Proof. Consider the following database.

ID	Sequence
S1	$\langle X, A, M, Y, A, M, D\rangle$
S2	$\langle Y, A, M, D, X, A, M\rangle$

Let $min\_sup = 1$ and $min\_conf = 0.5$. Then the CS-Key set contains the following patterns: $\langle A\rangle$:2, etc. The CS-Closed set contains the following patterns: $\langle X, A, M\rangle$:2, $\langle Y, A, M, D\rangle$:2, etc. The set of rules in Configuration (CS-Key, CS-Closed) includes the following: $\langle A\rangle \rightarrow \langle M\rangle$ (2,1.0), $\langle A\rangle \rightarrow \langle M, D\rangle$(2,1.0), etc. Using Property 4, the second rule listed above infers the first rule, hence the configuration is not tight. □

Lemma 4. *Configuration (LS-Key, LS-Closed) is not complete.*

Proof. Consider the following database.

ID	Sequence
S1	$\langle A, B, C\rangle$
S2	$\langle A\rangle$

Let $min\_sup = 1$ and $min\_conf = 0.5$. Then the LS-Key set contains the following patterns: $\langle A\rangle$:2, $\langle B\rangle$:1 and $\langle C\rangle$:1. The LS-Closed set contains the following patterns: $\langle A\rangle$:2, $\langle A,B\rangle$:1, $\langle A,B,C\rangle$:1. The set of rules in the configuration are: $\langle A\rangle \to \langle B\rangle$ (1,0.5), $\langle A\rangle \to \langle B,C\rangle$ (1,0.5) and $\langle B\rangle \to \langle C\rangle$ (1,1). Using Property 4, we do not have any rule in the mined set that infers the rule $\langle A,B\rangle \to \langle C\rangle$ (1,1) which is frequent and confident. Hence, the configuration is not complete. □

Proposition 5. *Configuration (CS-Key, CS-Closed), Configuration (CS-Key, LS-Closed), Configuration (LS-Key, CS-Closed) and Configuration (LS-Key, LS-Closed) are neither complete nor tight.*

Proof. Note that LS-Key ⊇ CS-Key. Also, LS-Closed ⊇ CS-Closed. Hence, from the definition of configuration, we have Configuration (LS-Key, LS-Closed) ⊇ Configuration (LS-Key, CS-Closed) ⊇ Configuration (CS-Key, CS-Closed). Also, Configuration (LS-Key, LS-Closed) ⊇ Configuration (CS-Key, LS-Closed) ⊇ Configuration (CS-Key, CS-Closed).

According to Lemma 3, Configuration (CS-Key, CS-Closed) is not tight. Hence, so are the other three configurations which are super-sets of Configuration (CS-Key, CS-Closed). Also, according to Lemma 4, Configuration (LS-Key, LS-Closed) is not complete. Hence, so are the other three configurations which are subsets of Configuration (LS-key, LS-Closed). □

Lemma 6. *Configuration (CS-Closed, CS-Closed) is not tight.*

Proof. Consider the following database.

ID	Sequence
S1	$\langle A,C,B,C,A,Z,B,C,D\rangle$
S2	$\langle A,Z,B,C,D,A,C,B,C\rangle$
S3	$\langle A,B\rangle$

Let $min\_sup = 2$ and $min\_conf = 0.5$. Then the CS-Closed set contains the following patterns: $\langle A,B\rangle$:3, $\langle A,C,B,C\rangle$:2, $\langle A,Z,B,C,D\rangle$:2, etc. The set of rules in Configuration (CS-Closed, CS-Closed) includes: $\langle A,B\rangle \to \langle C\rangle$ (2,0.67), $\langle A,B\rangle \to \langle C,D\rangle$ (2,0.67), etc. Using Property 4, the second rule listed above infers the first rule, hence the configuration is not tight. □

Lemma 7. *Configuration (LS-Closed, LS-Closed) is not complete.*

Proof. Consider the following database.

ID	Sequence
S1	$\langle A,B,C\rangle$
S2	$\langle A,B\rangle$

Let $min\_sup = 1$ and $min\_conf = 0.5$. Then the LS-Closed set contains the following patterns: $\langle A\rangle$:2, $\langle A,B\rangle$:2, and $\langle A,B,C\rangle$:1. The set of rules in Configuration (LS-Closed, LS-Closed) are the following: $\langle A\rangle \to \langle B\rangle$ (2,1.0), $\langle A\rangle \to \langle B,C\rangle$ (1,0.5), and $\langle A,B\rangle \to \langle C\rangle$ (1,0.5). Using Property 4, we do not have any rule in the mined set that infers the rule $\langle B\rangle \to \langle C\rangle$(1,0.5) which is frequent and confident. Hence, the configuration is not complete. □

Proposition 8. *Configuration (CS-Closed, CS-Closed), Configuration (CS-Closed, LS-Closed), Configuration (LS-Closed, CS-Closed) and Configuration (LS-Closed, LS-Closed) are neither complete nor tight.*

Proof. Note that LS-Closed ⊇ CS-Closed. Hence, from the definition of configuration, we have: Configuration (LS-Closed, LS-Closed) ⊇ Configuration (LS-Closed, CS-Closed) ⊇ Configuration (CS-Closed, CS-Closed). Also, Configuration (LS-Closed, LS-Closed) ⊇ Configuration (CS-Closed, LS-Closed) ⊇ Configuration (CS-Closed, CS-Closed).

According to Lemma 7, Configuration (CS-Closed, CS-Closed) is not tight. Hence, so are the other three configurations which are super-sets of Configuration (CS-Closed, CS-Closed). Also, according to Lemma 6, Configuration (LS-Closed, LS-Closed) is not complete. Hence, so are the other three configurations which are subsets of Configuration (LS-Closed, LS-Closed). □

4.2. Tight and complete set of non-redundant rules

Intuitively, the metarule (Key, Closed) matches closely the sufficient and necessary condition of rule inference described by Property 4. We want the mined rules to have shortest pre-conditions and longest post-conditions. However, no configuration in the (Key, Closed) family is complete and tight. We need to relax some constraints so that the set of rules become complete. We then need to tighten some other constraints so that the set of rules becomes tight.

We first relax the first input to the configuration to be a new pattern set defined below.

Definition 4.2 (*Prefix-Key*). A frequent pattern p is in Prefix-Key, iff there is no pattern p' where p' is a prefix of p and has the same support as p.

From Definitions 2.3 and 4.2, it can be noted that CS-Key is a subset of Prefix-Key. However, LS-Key and Prefix-Key are incomparable. We can then prove the following proposition.

Proposition 9. *Configuration (Prefix-Key, CS-Closed) is complete but not tight.*

Proof. We first prove that the configuration is complete. For every rule r that is frequent and confident, we would like to show that there exists $r' \in$ Configuration (Prefix-Key, CS-Closed), where r' infers r.

First, consider an arbitrary rule r of the form $pre \to post$, composed from two frequent patterns pre and pre++$post$. It can be inferred by another rule r_1 composed from $pre_1 \in$ Prefix-Key and pre++$post$. This is the case since there must be a prefix pre_1 of pre that belongs to Prefix-Key with the same support as pre. Composing pre_1 with pre++$post$, will form a rule $r_1 = pre_1 \to post_1$ where pre_1++$post_1 = pre$++$post$. From Property 4, r_1 infers r.

Second, $r_1 = pre_1 \rightarrow post_1$ can be inferred by another rule r_2, composed from pre_1 and $prepost_2 \in$ CS-Closed. This is the case since there must be a super-sequence of pre_1++$post_1$ which is in CS-Closed with the same support as pre_1++$post_1$. Let $prepost_2 \in$ CS-Closed $= pre_2$++$post_2$ such that $pre_1 \sqsubseteq pre_2$, $post_1 \sqsubseteq post_2$, pre_2 is as short as possible, and $post_2$ is as long as possible. Composing pre_1 with $prepost_2$ forms a rule $r_2 = pre_1 \rightarrow post_2$. From Property 4, r_2 infers r_1. Note that pre_1 is in Prefix-Key and $prepost_2$ is in CS-Closed.

Completeness now follows by transitivity of rule inference.

We next show that the configuration is not tight by means of a counter example. Consider the following database.

ID	Sequence
S1	$\langle A, Z, B, C, A, B, C, D, E, F\rangle$
S2	$\langle A, C, B, Z, A, B, C, D, E, F\rangle$
S3	$\langle A, B\rangle$
S4	$\langle A, Z\rangle$

Let $min\_sup = 1$ and $min\_conf = 0.5$. Then the Prefix-Key set contains the following patterns: $\langle A, B, C\rangle$:2, $\langle A, C\rangle$:2, $\langle A, Z, C\rangle$:2, etc. The CS-Closed set contains the following patterns: $\langle A, C, A, B, C, D, E, F\rangle$:2, $\langle A, Z, B, C, D, E, F\rangle$:2, etc. The set of rules in the configuration w.r.t. the above database includes: $\langle A, B, C\rangle \rightarrow \langle D, E, F\rangle$ (2,1.0), $\langle A, C\rangle \rightarrow \langle A, B, C, D, E, F\rangle$ (2,1.0), $\langle A, C\rangle \rightarrow \langle D, E, F\rangle$ (2,1.0), etc. Using Property 4, the second rule listed above infers the first rule, hence the configuration is not tight. $\square$

The set of redundant rules in Configuration (Prefix-Key,CS-Closed) is characterized by Lemma 10.

Lemma 10. *Let Configuration (Prefix-Key, CS-Closed) be denoted as CFG. The set of redundant rules in CFG denoted as REDUNDANT is the union of the following two sets*:

$\{r = pre \rightarrow post \in CFG \mid$ *there is* $r' = pre' \rightarrow post' \in CFG$ *having the same support and confidence*, $pre = pre'$ *and* $post' \sqsupset post\}$

and,

$\{r = pre \rightarrow post \in CFG \mid$ *there is* $r' = pre' \rightarrow post'$ *in CFG having the same support and confidence*, pre' *is a subsequence but not a prefix of* pre *and* pre'++$post' \sqsupseteq$ pre++$post\}$.

Proof. Suppose a rule $r = pre \rightarrow post$ in CFG is redundant and is inferred by another rule $r' = pre' \rightarrow post'$ also in CFG. From Property 4, we have: (1) $pre' \sqsubseteq pre$; (2) pre'++$post' \sqsupseteq pre$++$post$; (3) $sup(r) = sup(r')$; and (4) $\mathrm{conf}(r) = \mathrm{conf}(r')$. From 3 and 4, we have $sup(pre) = sup(pre')$.

Since $pre' \sqsubseteq pre$, $sup(pre') = sup(pre)$, and both pre and pre' are in Prefix-Key, there are two possible cases: (1) $pre' = pre$ and (2) pre' is a subsequence but not a prefix of pre. If $pre' = pre$, since r and r' are different, $post'$ must be a proper super-sequence of $post$. If the second case holds, condition (2) of Property 4 restated in the previous paragraph still holds, pre'++$post'$ must be a super-sequence of pre++$post$. $\square$

From the example described in the proof of Proposition 9, the rule $\langle A, C\rangle \rightarrow \langle D, E, F\rangle$, which is inferred by $\langle A, C\rangle \rightarrow \langle A, B, C, D, E, F\rangle$, belongs to the first redundant rule set described in Lemma 10. Also, the rule $\langle A, B, C\rangle \rightarrow \langle D, E, F\rangle$ (2,1.0), which is inferred by $\langle A, C\rangle \rightarrow \langle A, B, C, D, E, F\rangle$ (2,1.0), belongs to the second redundant rule set described in Lemma 10. From Proposition 9 and Lemma 10, we have the following theorem.

Theorem 11. *Configuration* (*Prefix-Key*, *CS-Closed*) $-$ *REDUNDANT is complete and tight.*

5. Compressed set of rules

Configuration(Prefix-Key,CS-Closed) $-$ REDUNDANT is the tight and complete set of non-redundant rules. Note that a pattern $X_k \in$ Prefix-Key is a frequent pattern. For every X_k in Prefix-Key, there is a closed pattern cp in LS-Closed where $X_k \in$ EQClass(cp, LS).

An equivalence class can be represented by either closed patterns or generators. The former set usually contains fewer members than the latter set. So we propose a mechanism to "compress" a non-redundant rule set by replacing the set of Prefix-Key generator premises with their corresponding set of LS-Closed closed patterns through a definition of a compression operation as follows.

Definition 5.1 (*Compressed rules*). We allow rules of the form $key(X) \rightarrow Y$ where $X \in$ LS-Closed. We call these the compressed rules. The meaning of a compressed rule denoted as $[key(X) \rightarrow Y]$ is $\{X_k \rightarrow Y | X_k \in$ EQClass $(X, LS, SeqDB) \wedge X_k \in$ Prefix-Key$\}$. Thus a sequence s satisfies $[key(X) \rightarrow Y]$ if for each $X_k \rightarrow Y \in [key(X) \rightarrow Y]$, we have $X_k \rightarrow Y\langle\langle s\rangle\rangle$. Note that s need not be $\in SeqDB$.

Then we can define the notion of a compressed rule set being sound and complete.

Definition 5.2 (*Sound and complete*). Consider a rule set R mined from $SeqDB$ w.r.t. min_sup and min_conf thresholds. A compressed rule set R_c is sound and complete for R iff the following hold: $R = \bigcup_{(key(X_k) \rightarrow Y_c) \in R_c} [key(X_k) \rightarrow Y_c]$ and $\forall r \in [key(X_k) \rightarrow Y_c]$. r has the same support and confidence as $key(X_k) \rightarrow Y_c$.

The compressed set of Configuration(Prefix-Key,CS-Closed) can be defined as:

Configuration-Key (LS-Closed, CS-Closed) $= \{key(pre) \rightarrow post | pre \in$ LS-Closed, $prepost' \in$ CS-Closed, $prepost' = sh$++$post$ where sh is the shortest prefix of $prepost'$ which is a super-sequence of $pre\}$.

The configuration is a sound and complete compression as stated in the following theorem.

Theorem 12. *Configuration-Key* (*LS-Closed*, *CS-Closed*) *is a sound and complete compression of Configuration* (*Prefix-Key*, *CS-Closed*).

Proof. Let us refer to Configuration (Prefix-Key,CS-Closed) and Configuration-Key (LS-Closed, CS-Closed) as CNF and CNFKey.

We first show that for an arbitrary rule $r = pre \rightarrow post$ in CNF, there exists a compressed rule $r' = key(cp) \rightarrow post$ having the same support and confidence in CNFKey, where cp is an LS-Closed pattern having the same projected database as pre. The above is the case since cp and pre has the same projected database, if we can extend pre to a pattern $Y = pre{+}{+}evs$ (where evs is an arbitrary series of events), we can also extend cp to a pattern $Z = cp{+}{+}evs$, where $sup(Y) = sup(Z)$. For both configurations CNF and CNFKey, the maximal extension is mined. Hence there should be a rule $r' = key(cp) \rightarrow post$ in CNFKey having the same support and confidence as r.

We next show that for all rules r in an arbitrary rule-set $[key(cp) \rightarrow post]$ in CNFKey, r is in CNF. $key(cp)$ gives a set of generators G in Prefix-Key with the same projected database as cp. Since cp has the same projected database as each member in G, if we can extend cp to a pattern $Y = cp{+}{+}evs$ (where evs is an arbitrary series of events), we can also extend $g \in G$ to a pattern $Z = g{+}{+}evs$, where $sup(Y) = sup(Z)$. Note that $[key(cp) \rightarrow post]$ can be empty. For both configurations CNF and CNFKey, the maximal extension is mined. Hence r should be in CNF and has the same support and confidence as $cp \rightarrow post$. □

Configuration-Key (LS-Closed,CS-Closed) can serve as a compression of the complete and tight set of non-redundant rules. To get back the complete and tight set of non-redundant rules from the compressed set one can compute the key( ) operation and subtract the set REDUNDANT from the resultant set. The compression hence is lossless. In this sense, we refer Configuration-Key (LS-Closed,CS-Closed) as the compressed non-redundant (CNR) rule set.

We would also like to mention the upper bound on the size of the compressed rule set. This is described in Corollary 13.

Corollary 13. *Given a non-redundant rule set R w.r.t. SeqDB, min_sup, and min_conf. The size of the compressed rule set* R_c *that is sound and complete for R is at most* $O(|LS-Closed| \times |CS-Closed|)$.

Proof. Given a rule $X_k \rightarrow Y_c$ in R. Let X_c be an LS-closed pattern of EqClass(X_k, LS). Then $X_k \rightarrow Y_c \in [key(X_c) \rightarrow Y_c]$. Thus the number of distinct X_c is limited by the number of LS-Closed patterns. Also, by construction of R, the number of distinct Y_c is limited by the number of CS-Closed patterns. Consequently, R_c is at most equal to $|LS\text{-}Closed| \times |CS\text{-}Closed|$. □

6. Mining algorithm

To generate the CNR set of rules, i.e., Configuration-Key (LS-Closed, CS-Closed), one must first mine the patterns. There are existing algorithms to mine CS-Closed set, e.g., BIDE [7]. However, there is no algorithm in existing literature to mine LS-Closed.

Fortunately, BIDE, in effect employs a search space pruning strategy that prune all search sub-spaces that contain patterns not in LS-Closed set. A pattern that is not pruned is subjected to an online tests (i.e., while closed patterns are being generated) to check whether it is in CS-Closed. With modification, rather than discarding patterns that are in LS-Closed but not in CS-Closed, we also print these patterns out while flagging them as belonging to LS-Closed exclusively. Hence, we have an implementation of an algorithm that can concurrently mine both LS- and CS-Closed patterns. The details and proofs that this modification does indeed produce LS- and CS-Closed patterns are available in Appendix A.

Our proposed algorithm (CNR rule mining algorithm) is shown in Fig. 2. The algorithm is based on the definition of Configuration-Key (LS-Closed,CS-Closed). The algorithm starts (lines 1–3) by mining the LS- and CS-Closed sets concurrently. For each pattern p in LS-Closed, it then tries to find related patterns p' in CS-Closed which are its super-sequences (lines 4–5). Next, for each related pattern p', we try to form a rule $rn = p \rightarrow post$ where $p' = sh{+}{+}post$ and sh is the shortest prefix of p' containing p (lines 7–9). Following Property 1, the support of $p{+}{+}post$ is equal to the maximum support of a closed pattern that is a super-sequence of it. Hence, to get the support of $p \rightarrow post$, we perform such checks and update our temporary rule set $Rules'$ in lines 10–15. When the support of a generated rule is not correct, we eventually either update the support or throw the rule away. If a rule with the same format (i.e., $p \rightarrow post$) and correct support is found, we update the support value of the previously generated rule having incorrect support (line 11). Otherwise, we discard the previously generated rule, since according to Definition 4.1, a rule in a configuration must have the correct support and syntactic characterization (line 13). At the end of each iteration (before line 16), the temporary rule set $Rules'$ contains the set of compressed rules having p as premise with correct support and confidence. At line 16, we add the rules in $Rules'$ to the output rule set $Rules$. After iterating for all possible patterns in LS-Closed, $Rules$ will correspond to the set Configuration-Key (LS-Closed,CS-Closed) which is then output.

We would like to measure the performance benefit of mining a set of non-redundant rules over mining a full-set of significant rules. For comparison purpose, Fig. 3 describes an algorithm mining a full-set of significant rules based on the description in [8]. The algorithm first computes the set of all frequent patterns. We use PrefixSpan [3][1] to mine for all frequent patterns.[2] For each frequent pattern f of length l, we can form $l-1$ rules. Each rule is of the form $pre \rightarrow post$, where pre is a prefix of f and $pre{+}{+}post = f$.

[1] PrefixSpan has a similar depth-first mining architecture as BIDE [7] used by our non-redundant rule mining algorithm.

[2] In case too much memory is needed to load all frequent patterns at line 1, we load the patterns in batches and process them accordingly.

```
Procedure Mine Compressed Non-Redundant Rules
Inputs: $SeqDB$:  Sequence Database ;
$min\_sup$, $min\_conf$:  Minimum support and confidence thresholds
Outputs: $Rules$:  Compressed set of non-redundant rules
Method:
1:  Let $LS = \{\}$, $CS = \{\}$, $Rules = \{\}$
2:  Concurrently mine LS- and CS-Closed with support $\geq min\_sup$
3:  Initialize $LS$ and $CS$ to LS- and CS-Closed accordingly
4:  For each pattern $p \in LS$
5:      Let $Related = \{p' \in CS$, where $p \sqsubset p'$ and
                    $(min\_conf \leq sup(p')/sup(p) \leq 1)\}$
6:      Let $Rules' = \{\}$
7:      For each pattern $p' \in Related$
8:          Let $post =$  $px$, where $sx{+\!+}px = p'$, $sx \sqsupseteq p$, and
                    $sx$ is as short as possible
9:          Let $rn = p \rightarrow post$, $nsup = sup(p')$ and $nconf = sup(p')/sup(p)$
10:         If $\exists\ r(s,c) \in Rules'.\ r$.Post $= rn$.Post
11:             If $(nsup > s)$ Replace $r(s,c)$ in $Rules'$ with $r(nsup,nconf)$
12:         Else If $\exists\ r(s,c) \in Rules'.\ r$.Post $\sqsubset rn$.Post
13:             If $(nsup > s)$ Remove $r(s,c)$ from $Rules'$
14:             Add $rn(nsup,nconf)$ to $Rules'$
15:         Else Add $rn(nsup,nconf)$ to $Rules'$
16:     $Rules = Rules \cup Rules'$
Output  $Rules$
```

Fig. 2. CNR algorithm.

7. Performance study

We perform a performance study using the synthetic data generator provided by IBM which was also used in [1,6]. We modify the data generator to ensure generation of sequences of events (i.e., all transactions are of size 1). We also consider a real sequence dataset from commonly used benchmark in software engineering field.

Experiments were performed on a Fujitsu E4010 laptop with Intel Mobile 1.6 GHz and 512 MB main memory, running Windows XP Professional. Algorithms were written using Visual C#.Net running under .Net Framework 1.1 compiled with release mode using Visual Studio.Net 2003.

The experiment results for dataset D5C20N10S20 are shown in Figs. 4 and 5. The parameters D, C, N and S correspond to the number of sequences (in 1000's), the average number of events per sequence, the number of different events (in 1000's) and the average number of events in the maximal sequences. "Full" corresponds to mining a full set of frequent and confident sequential rules, while "CNR" corresponds to mining a compressed set of non-redundant rules. Similar to other study in mining from sequences [6,7] we use a low support threshold to test for scalability. Also, mining for rules with significant confidence at low support thresholds is particularly valuable for diverse dataset like the video shop example given in Section 1. The experiment results with the real dataset shows that performance speed up can be obtained in both high and low support thresholds and is necessary to analyze real data.

The study shows large improvements in both runtime and compactness of mined rules over mining a full set of sequential rules. Runtime was improved up to 5598 *times*! The number of rules was reduced up to 8583 *times*!

```
Procedure Mine All Significant Rules
Inputs: $SeqDB$:  Sequence Database ;
$min\_sup$, $min\_conf$:  Minimum support and confidence thresholds
Outputs:  All significant rules
Method:
1: Let $Freq$ = All patterns with support $\geq min\_sup$.
2: For each pattern $f \in Freq$
3:   For each prefix $pre$ of $f$
4:     Let $post = px$, where $pre{+}{+}px = f$
5:     Let $rn = pre \rightarrow post$, $nsup = sup(f)$ and $nconf = sup(f)/sup(pre)$
6:     If $(nconf \geq min\_conf)$
7:       Output $rn(nsup, nconf)$
```

Fig. 3. Algorithm mining all significant rules.

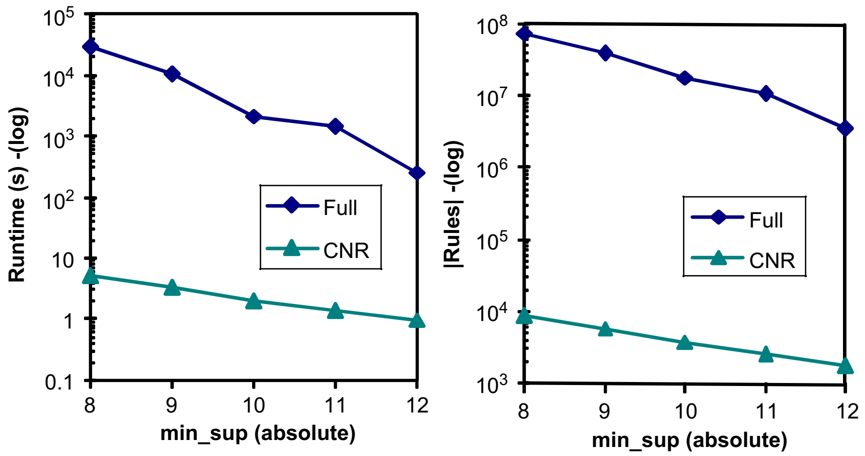

Fig. 4. Varying *min_sup* at *min_conf* = 50% for D5C20N10S20 dataset.

Recently, there have been active interests in analyzing program traces [21]. We generate traces from a simple Traffic alert and Collision Avoidance System (TCAS) from Siemens Test Suite [22] used as one of the benchmarks for research in error localization (e.g., [23]). The test suite comes with 1578 correct test cases. We run the test cases and obtain 1578 traces. Each trace is treated as a sequence. The sequences are of average length of 61 and maximum length of 97. It contains 106 different events—the events are the line numbers of the statements being executed. We call this dataset as TCAS dataset.

Mining rules from program execution traces can shed light on implicit rules that govern the behavior of program or which are made by programmers. Since the analysis is based on traces rather than code, one will not face the problem of infeasible paths [24], also dynamic inputs and environment will be taken into consideration and the analysis is not restricted to cases where source code is available (i.e., third-party binary code, network events, etc.). These rules can potentially be used for detection of abnormal behavior either corresponding to software bugs, anomalies or intrusion (cf., [25]). We leave these potential case studies for future work and focus more on performance issues.

In the experiments with TCAS dataset, we did not mine the full set of rules as the number of rules are huge and not minable even at support level of 100%. The traces share many similarities (the longest rule of support 100% is of length 28). The result for mining CNR rule set is shown in Fig. 6. It shows that the non-redundant rule mining algorithm can work in a real application setting where the full-set rule mining algorithm fails due to scalability issues. This shows a major benefit of mining CNR rules.

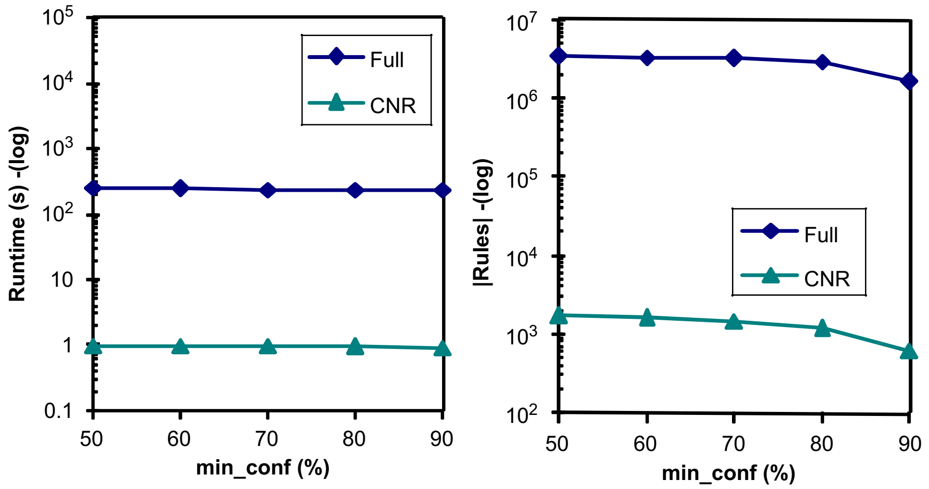

Fig. 5. Varying *min_conf* at *min_sup* (absolute) = 12 for D5C20N10S20 dataset.

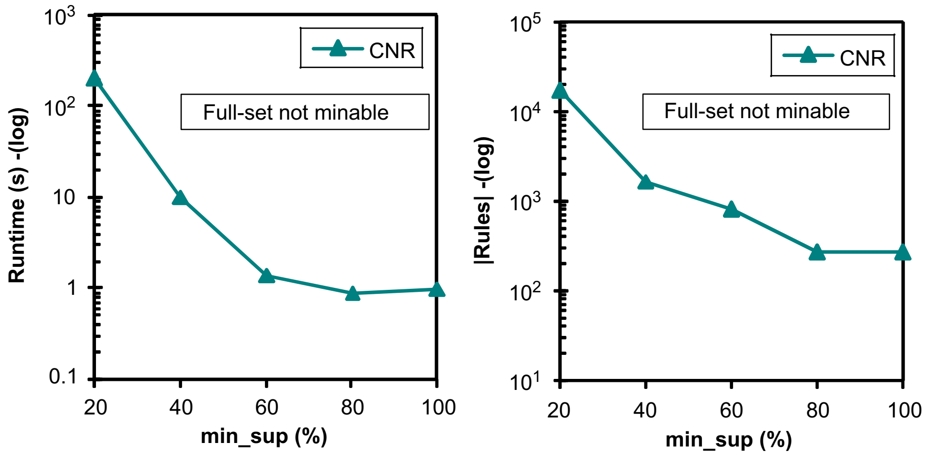

Fig. 6. Varying *min_sup* (in %) at *min_conf* = 50% for TCAS dataset.

8. Related work

We discuss three areas of research related to our work.

Non-redundant association rule mining and non-derivable itemset mining: Zaki and Hsiao mined a non-redundant set of association rules [26] (see also [27]). Different from the usual association rules, *sequential ordering* is important in a sequential rule setting. Sequential rules and association rules are formed from different types of patterns: sequential patterns vs. itemsets. $\langle A,B\rangle$ and $\langle B,A\rangle$ are regarded as different sequential patterns but the same itemset. Consequently, their mining processes differ significantly.

The association rule generation technique of Zaki and Hsiao hinges on the following *equivalence property*: two rules $r_1 = X_1 \rightarrow X_2$ and $r_2 = Y_1 \rightarrow Y_2$ are considered equivalent iff $c_{it}(X_1) = c_{it}(Y_1)$ and $c_{it}(X_2) = c_{it}(Y_2)$, where $c_{it}(X)$ denotes X's representative closed itemset. X's representative closed itemset is a maximal super-set of X supported by the same sequences in the database. Given a set of equivalent rules, only its representative rules need to be reported; the rest are considered redundant. This equivalence property *no longer holds in a sequential rule setting*, as illustrated below.

Consider a sequence database consisting of: $S1 : \langle A,C,B\rangle$ and $S2 : \langle C\rangle$. Consider two rules $r_1 = \langle A\rangle \rightarrow \langle C\rangle$ and $r_2 = \langle A,B\rangle \rightarrow \langle C\rangle$. These two rules are considered equivalent in an association-rule setting. However, they are *not* when *sequential ordering* is taken into consideration: as sequential rules, r_1 has 100% confidence and 50% support, whereas r_2 is not even exhibited in the database.

Hence, the approaches to mine a non-redundant set of sequential rules and association rules are different from each other, much as sequential pattern mining is different from frequent itemset mining.

A related study by Calders et al. [28] discuss non-derivable frequent itemsets. A frequent itemset is non-derivable if its *support value* cannot be inferred from the support of one or more of its subsets. In this study, different from the study by Calders we focus on both rules and sequential data. Also, we do not consider *inference of support value* rather *inference of the rule* itself.

Mining closed sequential pattern and generators: The pioneer work in closed-pattern mining was CloSpan by Yan et al. [6]. It was further extended by Wang and Han [7]. Gao et al. and Lo et al. mines sequential generators [18,19]. Compared to mining a full set of frequent patterns proposed in [1], closed sequential pattern mining and sequential generator mining run much faster and potentially reduces the number of mined patterns.

Sequential rules extend the usability and expressiveness of patterns beyond the understanding of sequential data. A mined rule represents a *constraint* that its premise is followed by its consequent in sequences. Furthermore, the interestingness of a rule is measured by both support and confidence. The notion of confidence is useful especially when the support threshold specified is low. Hence, rules are potentially useful for detecting and filtering anomalies which violate the corresponding constraints. They have potential application in detecting errors, intrusions, bugs, etc. Mining rule-like sequencing constraints from sequential data has also been shown useful in medicine (e.g., [10]) and software engineering (e.g., [11–13]) domains.

In this paper, the question *how a set of non-redundant rules can be generated from compact representative patterns*, is comprehensively addressed. Issues pertaining to generating a non-redundant set of rules are *orthogonal* to that pertaining to generating a closed set of patterns or a set of sequential generators. We introduce the concept of rule inference to generate a non-redundant rule set in which all frequent and confident rules are either reported or inferred by some reported rules.

Generation of sequential rules: Spiliopoulou has proposed generating a full set of sequential rules from a full set of sequential patterns [8]. Generation of a full set of rules is often not scalable. The number of frequent patterns is exponential to the maximum pattern length, and for every frequent sequential pattern of length l, possibly $\mathcal{O}(2^l)$ rules can be generated. Spiliopoulou added a post-mining filtering step to remove some redundant rules. Unfortunately, the number of intermediate patterns and rules can be very large.

Recently, studies in [10–13,29] mine temporal rules, outlier detection rules, sequential classification rules, progressive confident rules and recurrent rules, respectively. These rules can be considered as variants of sequential rules; they add or remove some constraint or information from the sequential rules. In this study, we focus on the classical sequential rules. More importantly, different from our study, none of the above studies consider redundancy based on rule inference. Aside from these more general differences, other specific differences are described in the following paragraphs.

The study in [11] only mines temporal rules of length 2; rules of longer length are formed by a simple concatenation of length-2 rules. Some length-(>2) rules might be missed or introduced although not having enough significance. The study in [12] mines outlier detection rules; however, the confidence and support values are only approximated. Also, there is no guarantee that a complete set of rules are mined. Different from the above studies, we guarantee completeness, correctness of reported support and confidence values and also tightness.

The studies in [10,29] mine two different variants of classification rules. A mined rule pre-condition is a series of events, while the post-condition is a classification decision or state. Different from the above, we consider a different and more general problem where a rule can have a post-condition composed of multiple events.

Among these five studies, nearest to our study is the study in [13] which mines long recurrent rules by actively removing those *shorter rules which are sub-sequences* of the longer rules. In [13], the resultant set of rules will be more compact, but there will be *no semantic, but only syntactic* relationship between the smaller set of rules and the original set of rules. Using the set of mined rules as a composite filter, replacing a full-set of rules with the non-redundant set of rules *may potentially* impact the accuracy of the filter. Since these studies mined *different rules and consider different scenarios*, we focus our comparative study on the classical sequential rules originally proposed by Spiliopoulou [8]. In the future, we are looking into extending the study further to address non-redundant rule mining based on classical rule inference to the above studies.

9. Discussion

In this section, we discuss: (1) uniqueness of a tight and complete set of non-redundant rules, (2) more complex rule inference strategy.

Uniqueness of a tight and complete set of rules: An interesting question is given a sequence database and a minimum support threshold, is there *only one unique* or are *there multiple* tight and complete sets of non-redundant rules? The following theorem and lemma shows that given a sequence database and a minimum support threshold, there is only one unique tight and complete set of non-redundant rules.

Theorem 14. *Given a sequence database and a minimum support threshold, there is only one unique tight and complete set of non-redundant rules.*

Proof. Assume for contradiction that $\exists RSet' \neq RSet$ and $RSet$ and $RSet'$ are tight and complete sets of non-redundant rules mined from $SeqDB$ at min_sup. There must exists a rule r_1 where $r_1 \in RSet'$ and $r_1 \notin RSet$. Since $RSet$ is complete $\exists r_2 \in Rset$. $r_2 \rightarrow r_1$. Since $RSet'$ is tight, $\nexists r_2 \in RSet'$. Also since $RSet'$ is complete $\exists r_3 \in RSet'$. $r_3 \rightarrow r_2$. However, due to transitivity property of rule inference, $r_3 \rightarrow r_1$. This is a contradiction since we assume that $RSet'$ is tight. Note that, due to Property 4, $r_1 = r_3$ is impossible unless $r_1 = r_2$. □

More complex rule inference strategy: In this study, we consider a rule to be redundant if there exists another mined rule that infers it. We guarantee that the resultant non-redundant rule set to be tight and complete. Another interesting study is to consider more complex rule inference strategy involving inference by a set of rules: multiple mined rules can infer another mined rule which can then be rendered removed. The issue of potentially multiple solution sets (i.e., non-uniqueness of resultant non-redundant rule set) need to be addressed accordingly.

We leave this interesting direction of study for future work. In this work, we see that even with the current redundancy inference strategy, there is a huge reduction in the size of mined rules and improvement in the mining speed at both high-and-low support thresholds.

10. Conclusion and future work

In this paper, we propose and characterize a *non-redundant* set of sequential rules. A mined rule is redundant if it can be inferred by another mined rule. We base our investigation and characterization on past studies on compact representative sets of sequential patterns. In particular we address the following questions not studied before in the literature: Can a non-redundant set of sequential rules be obtained from compact representative sequential patterns? What types of compact representative patterns need to be mined to form non-redundant rules? What do we mean by a non-redundant set of rules? Can we characterize the non-redundant set of rules? How to use representative patterns to form non-redundant rules? How much effort is needed to obtain a non-redundant set of rules from compact representative patterns? Can we design an efficient algorithm to obtain a non-redundant set of rules from patterns?

To answer the above questions, we investigate various configuration of rule sets based on composition of four different pattern sets. We analyze these rule-sets based on the properties of completeness (i.e., all rules can be inferred) and tightness (i.e., no mined rules are redundant). We find that Configuration (Prefix-Key,CS-Closed) $-$ REDUNDANT is the tight and complete set of non-redundant sequential rules. Additionally, we propose and characterize a compressed set of non-redundant rules. A mining algorithm has been proposed and developed to mine this compressed set of rules. A performance study has been performed to evaluate the benefit of mining CNR rule set. The study shows large improvements in both runtime and compactness of mined rules over mining a full set of sequential rules. Runtime was improved up to 5598 *times*! The number of rules was reduced up to 8583 *times*!

As a future work, we plan to investigate the following areas. First, we would like to try to improve further the efficiency of the mining algorithm. Also, it is the case that if some events A and B occur frequently in the background, then rules $A \rightarrow B$ and/or $B \rightarrow A$ will both have high support and high confidence. Yet these rules are coincidental and do not indicate any meaningful sequential relationship between A and B. We plan to consider generating non-redundant set of rules while avoiding producing such purely coincidental rule. For example, by performing a hypergeometric test (Fisher's exact test) on the co-occurrence of A and B. We believe the incorporation of such a test may be an interesting direction for future work.

Furthermore, investigation of application of non-redundant set of sequential rules to tasks such as web-log analysis, software engineering, etc. will also be an interesting direction we plan to explore. We believe the efficiency of non-redundant sequential rule mining and the compactness of mined rules will facilitate applications of sequential rules to more application domains.

Appendix A. Concurrently mining CS- and LS-Closed

To understand how BIDE [7] can be modified to concurrently mine both CS- and LS-Closed sets, in this section, we first describe some terminologies mentioned in BIDE's paper [7], present some lemmas and relate these to how BIDE's algorithm can be modified.

Definition A.1 (*First instance*). Given a sequence S which contains a single event pattern $\langle e_1 \rangle$, the prefix of S to the first appearance of the event e_1 in S is called the first instance of pattern $\langle e_1 \rangle$ in S. Recursively, we can define the first instance of an $(i+1)$-event pattern $\langle e_1, e_2, \ldots, e_i, e_{i+1} \rangle$ from the first instance of the i-event pattern $\langle e_1, e_2, \ldots, e_i \rangle$ (where $i \geqslant 1$) as the prefix of S to the first appearance of event e_{i+1} which also occurs after the first instance of the i-event pattern $\langle e_1, e_2, \ldots, e_i \rangle$. For example, the first instance of the prefix sequence $\langle A, B \rangle$ in sequence $\langle C, A, A, B, C \rangle$ is $\langle C, A, A, B \rangle$.

Definition A.2 (*i-th last-in-first appearance*). For an input sequence S containing a pattern $S_p = \langle e_1, e_2, \ldots, e_n \rangle$, the i-th last-in-first appearance w.r.t. the pattern S_p in S is denoted as LF_i and defined recursively as: (1) if $i = n$, it is the last appearance of e_i in the first instance of pattern S_p in S; (2) if $1 \leqslant i < n$, it is the last appearance of e_i in the first instance of the pattern S_p in S while LF_i must appear before LF_{i+1}. For example, if $S = \langle C, A, A, B, C \rangle$ and $Sp = \langle C, A, C \rangle$, the 2nd last-in-first appearance w.r.t. pattern S_p in S is the second A in S.

Definition A.3 (*i-th semi-maximum period*). For an input sequence S containing a pattern $S_p = \langle e1, e2, \ldots, e_n \rangle$, the i-th semi-maximum period of the pattern S_p in S is defined as: (1) if $1 < i \leqslant n$, it is the piece of sequence between the end of the first instance of pattern $\langle e_1, e_2, \ldots, e_{i-1} \rangle$ in S (exclusive) and the i-th last-in-first appearance w.r.t. pattern S_p (exclusive); (2) if $i = 1$, it is the piece of sequence in S locating before the 1st last-in-first appearance w.r.t. pattern S_p. For example, if $S = \langle A, B, C, B \rangle$ and the pattern $S_p = \langle A, C \rangle$, the 2nd semi-maximum period of prefix $\langle A, C \rangle$ in S is $\langle B \rangle$, while the 1st semi-maximum period of pattern $\langle A, C \rangle$ in S is an empty string.

Lemma 15. *P is in LS-Closed if and only if there does not exist an e and i, where event e appears in the i-th semi-maximum periods of P for every S* $\in$ *SeqDB.*

Proof. *The left to right direction*: We first show that if P is in LS-Closed, then there is no e and i, where event e is in the i-th semi-maximum periods of P for every $S \in SeqDB$. Taking the contrapositive of the above statement we have if there is an e and i, where event e is in the i-th semi-maximum periods of P for every $S \in SeqDB$, then P is not in LS-Closed.

Suppose there is an event e which is in the i-th semi-maximum period of P for every $S \in SeqDB$, we can then form a longer pattern P' by inserting e between event $(i-1)$ and (i) of P (if $i > 1$) or by pre-pending e before P (if $i = 1$) which will have the same support as P. From the definition of semi-maximum period, first instances of P' and P in $SeqDB$ will be the same. Hence, P and P' have the same projected database. Since there exists a P' which is a super-sequence of P having the same projected database, P is not in LS-Closed. This is a contradiction. We have proven the *left to right direction* of the lemma.

The right to left direction: We next need to show that if there is no e and i, where event e is in the i-th semi-maximum period of P for every $S \in SeqDB$, P will be in LS-Closed. Again, taking the contrapositive, the above statement is equivalent to: if P is not in LS-Closed then there exists an e and i, where event e is in the i-th semi-maximum period of P for every $S \in SeqDB$.

Suppose P is not in LS-Closed, this means that there exists a longer pattern P', where P' is a super-sequence of P, the length of P' is one event longer than P and they have the same projected database. It must be the case then that there exists two shorter patterns X and Y (with X possibly empty), where:

$$P = X + +e2 + +Y$$
$$P' = X + +e + +e2 + +Y$$

Since P and P' have the same projected database, for every sequence S in $SeqDB$, the first instance of P' in each sequence S which is a super-sequence of P' in $SeqDB$ will also be the first instance of P. Let i be the length of pattern X. From the above, event e must occur between the first instance of X (exclusive) and the $(i+1)$-st last-in-first appearance w.r.t. to P (exclusive) for every sequence S in $SeqDB$. From the definition of semi-maximum period, e must be in the $(i+1)$-st semi-maximum period of P for every $S \in SeqDB$. We have proven the *right to left direction* of the lemma. □

Lemma 16. *If P and P′ have the same projected database and P′ is a super-sequence of P, then for an arbitrary series of events evs, P++evs will not be in LS-Closed.*

BIDE employs the search space pruning strategy called backscan pruning: Let *evs* be an arbitrary series of events, if a pattern P has an event e appearing in each of its i-th semi-maximum period for all sequence S in $SeqDB$ than P as well as *P++evs* are not in CS-Closed. Lemma 15 guarantees that any pattern not pruned by the backscan pruning strategy must be in LS-Closed. Lemma 16 guarantees that there is no point in extending pattern P if it has been pruned by the backscan pruning strategy.

Using the above two lemmas, one can continue to cut the search space by using the backscan pruning of BIDE. BIDE employs an online check to see whether a pattern which is not pruned in CS-Closed which is called the BIDE closure checking scheme. We can distinguish members of LS-Closed that is not a member of CS-Closed by the result of this check. The runtime of modified BIDE is similar to the original BIDE since we cut the same search space as BIDE, i.e., search space containing those patterns which are not in LS-Closed.

References

[1] R. Agrawal, R. Srikant, Mining sequential patterns, in: Proceedings of IEEE International Conference on Data Engineering, 1995, pp. 3–14.

[2] M. Zaki, SPADE: An efficient algorithm for mining frequent sequences, Machine Learning 42 (2001) 31–60.

[3] J. Pei, J. Han, B. Mortazavi-Asl, Q. Chen, U. Dayal, M. Hsu, PrefixSpan: mining sequential patterns efficiently by prefix-projected pattern growth, in: Proceedings of IEEE International Conference on Data Engineering, 2001, pp. 215–226.

[4] J. Han, J. Wang, Y. Lu, P. Tzvetkov, Mining top-k frequent closed patterns without minimum support, in: Proceedings of IEEE International Conference on Data Mining, 2002, pp. 211–218.

[5] J. Ayres, J. Gehrke, T. Yiu, J. Flannick, Sequential pattern mining using a bitmap representation, in: Proceedings of SIGKDD Conference on Knowledge Discovery and Data Mining, 2002, pp. 429–435.

[6] X. Yan, J. Han, R. Afshar, Clospan: mining closed sequential patterns in large datasets, in: Proceedings of SIAM International Conference on Data Mining, 2003.

[7] J. Wang, J. Han, BIDE: efficient mining of frequent closed sequences, in: Proceedings of IEEE International Conference on Data Engineering, 2004, pp. 79–90.

[8] M. Spiliopoulou, Managing interesting rules in sequence mining, in: Proceedings of European Conference on Principles of Data Mining and Knowledge Discovery, 1999, pp. 554–560.

[9] R. Agrawal, R. Srikant, Fast algorithms for mining association rules, in: Proceedings of International Conference on Very Large Data Bases, 1994, pp. 487–499.

[10] M. Zhang, W. Hsu, M.-L. Lee, Mining progressive confident rules, in: Proceedings of SIGKDD Conference on Knowledge Discovery and Data Mining, 2006, pp. 803–808.

[11] J. Yang, D. Evans, D. Bhardwaj, T. Bhat, M. Das, Perracotta: mining temporal API rules from imperfect traces, in: Proceedings of International Conference on Software Engineering, 2006, pp. 282–291.

[12] D. Lo, S.-C. Khoo, SMArTIC: toward building an accurate, robust and scalable specification miner, in: Proceedings of SIGSOFT Symposium on the Foundations of Software Engineering, 2006, pp. 265–275.

[13] D. Lo, S.-C. Khoo, C. Liu, Efficient mining of recurrent rules from a sequence database, in: Proceedings of International Conference on Database Systems for Advanced Applications, 2008, pp. 67–83.

[14] Windows Driver Kit: driver development tools—SpinLock ⟨http://msdn.microsoft.com/en-us/library/aa469109.aspx⟩.

[15] F. Masseglia, F. Cathala, P. Poncelet, The PSP approach for mining sequential patterns, in: Proceedings of European Conference on Principles of Data Mining and Knowledge Discovery, 1998, pp. 176–184.

[16] R. Srikant, R. Agrawal, Mining sequential patterns: generalizations and performance improvements, in: Proceedings of International Conference on Extending Database Technology, 1996, pp. 25–29.

[17] J. Han, J. Pei, B. Mortazavi-Asl, Q. Chen, U. Dayal, M. Hsu, FreeSpan: frequent pattern-projected sequential pattern mining, in: Proceedings of SIGKDD Conference on Knowledge Discovery and Data Mining, 2000, pp. 355–359.

[18] D. Lo, S.-C. Khoo, J. Li, Mining and ranking generators of sequential patterns, in: Proceedings of SIAM International Conference on Data Mining, 2008, pp. 553–564.

[19] C. Gao, J. Wang, Y. He, L. Zhou, Efficient mining of frequent sequence generators, in: Proceedings of International Conference on World Wide Web, 2008, pp. 1051–1052.

[20] J. Han, M. Kamber, Data Mining Concepts and Techniques, Morgan Kaufmann, Los Altos, CA, 2001.

[21] Workshop on Dynamic Analysis ⟨http://www.cs.wisc.edu/woda-2008/⟩. Last accessed 24 February 2009.

[22] M. Hutchins, H. Foster, T. Goradia, T. Ostrand, Experiments of the effectiveness of dataflow- and controlflow-based test adequacy criteria, in: Proceedings of International Conference on Software Engineering, 1994, pp. 191–200.
[23] H. Cleve, A. Zeller, Locating causes of program failures, in: Proceedings of International Conference on Software Engineering, 2005, pp. 342–351.
[24] B. N, S. M, M.-C. Gaudel, S.-D. Gouraud, A machine learning approach for statistical software testing, in: Proceedings of International Joint Conferences on Artificial Intelligence, 2007, pp. 2274–2279.
[25] Z. Li, Y. Zhou, PR-Miner: automatically extracting implicit programming rules and detecting violations in large software code, in: Proceedings of Joint Meeting of European Software Engineering Conference and SIGSOFT Symposium on the Foundations of Software Engineering, 2005, pp. 306–315.
[26] M. Zaki, C. Hsiao, CHARM: an efficient algorithm for closed itemset mining, in: Proceedings of SIAM International Conference on Data Mining, 2002.
[27] M. Zaki, Mining non-redundant association rules, Data Mining and Knowledge Discovery 9 (3) (2004) 223–248.
[28] T. Calders, B. Goethals, Mining all non-derivable frequent itemsets, in: Proceedings of European Conference on Principles of Data Mining and Knowledge Discovery, 2002, pp. 74–85.
[29] E. Baralis, S. Chiusano, R. Dutto, Applying sequential rules to protein localization prediction, Computer and Mathematics with Applications 55 (5) (2008) 867–878.