Fournier-Viger, P., Faghihi, U., Nkambou, R., Mephu Nguifo, E. (2012).
CMRules: Mining Sequential Rules Common to Several Sequences.
Knowledge-based Systems, Elsevier, 25(1): 63-76.

CMRules: Mining Sequential Rules Common to Several Sequences

Philippe Fournier-Viger[1], Usef Faghihi[1], Roger Nkambou[1], and Engelbert Mephu Nguifo[2]

[1]Department of Computer Sciences, University of Quebec in Montreal,
201, avenue du Président-Kennedy, Montréal, Canada
{fournier-viger.philippe, nkambou.roger}@uqam.ca
[2]Department of Mathematics and Computer Sciences, Université Blaise-Pascal Clermont 2,
BP 125, 63173 Aubière, cedex, France
engelbert.mephu_nguifo@univ-bpclermont.fr

Abstract— Sequential rule mining is an important data mining task with wide applications. However, current algorithms for discovering sequential rules common to several sequences use very restrictive definitions of sequential rules, which make them unable to recognize that similar rules can describe a same phenomenon. This can have many undesirable effects such as (1) similar rules that are rated differently, (2) rules that are not found because they are considered uninteresting when taken individually, (3) and rules that are too specific, which makes them less likely to be used for making predictions. In this paper, we address these problems by proposing a more general form of sequential rules such that items in the antecedent and in the consequent of each rule are unordered. We propose an algorithm named CMRules, for mining them. The algorithm proceeds by first finding association rules to prune the search space for items that occur jointly in many sequences. Then it eliminates association rules that do not meet the minimum confidence and support thresholds according to the time ordering. We evaluate the performance of CMRules in three different ways. First, we provide an analysis of its time complexity. Second, we compare its performance (execution time, memory usage and scalability) with an adaptation of an algorithm from the literature that we name CMDeo. For this comparison, three real-life public datasets representing three types of data and having varied characteristics. Results show that for some datasets CMRules is faster and has a better scalability for low support thresholds. Lastly, we report a successful application of the algorithm in a tutoring agent.

Keywords— Sequential rule mining, association rule mining, sequence database, temporal rules

1 INTRODUCTION

Nowadays, huge amounts of temporal information are stored in databases (e.g. stock market data, biological data, patient hospital records and customer data). Discovering temporal relationships in such databases is important in many domains, as it provides a better understanding of the data, and sets a basis for making predictions. For example, in international trade, one could be interested in discovering relations between the appreciations of currencies to make trade decisions. Various methods have been proposed for mining temporal relations in temporal data (see for example [12], for a survey).

In the field of data mining, one of the most popular techniques for discovering temporal relations in discrete time series is *sequential pattern mining* (SPM) [2, 20, 21, 22]. SPM algorithms find sequential patterns, subsequences that appear frequently in a sequence database (a set of sequences). However, knowing that a sequence appear frequently in a database is not sufficient for making prediction. For example, it is possible that an event *c* appears frequently after some events *a* and *b* but that there are also many cases where *a* and *b* are not followed by *c*. In this case, predicting that c will occur after *ab* according to a sequential pattern *abc* could be a huge mistake. Thus, to make predictions, it is desirable to have patterns that indicate how many times *c* appears after *ab* and how many times it does not. But adding this information to sequential patterns cannot be done easily as sequential patterns are lists of events that can contain several events – not just three, as in the previous example– and current SPM algorithms have just not been designed for this.

The alternative to sequential pattern mining that addresses the problem of prediction is sequential rule mining [4, 5,

8, 9, 11, 13, 34, 35, 36]. A sequential rule (also called episode rule, temporal rule or prediction rule) indicates that if some event(s) occurred, some other event(s) are likely to occur with a given confidence or probability. Sequential rule mining has been applied in several domains such as stock market analysis [4, 11], weather observation [8], e-learning [6, 16] and drought management [5, 9].

The most famous approach to sequential rule mining is that of Mannila et al. [13] and other researchers afterward that aim to discover partially ordered sets of events appearing frequently within a time window in a sequence of events. Given these "frequent episodes", a trivial algorithm can derive all sequential rules respecting a minimal confidence and support [13]. These rules are of the form X⇒Y, where X and Y are two sets of events, and are interpreted as "if event(s) X appears, event(s) Y are likely to occur with a given confidence afterward". However, this approach can only discover rules in a single sequence of events. Other works that extract sequential rules from a single sequence of events are the algorithms of Hamilton & Karimi [8], Hsieh et al. [11] and Deogun & Jiang [5], which respectively discover rules between several events and a single event, between two events, and between several events.

Contrarily to these works that discover rules in a single sequence of events, a few algorithms have been designed to mine sequential rules in several sequences [4, 9, 14, 34, 35, 36]. For example, Das et al. [4] proposed an algorithm to discover rules where the left part of a rule can have multiple events, yet the right part contains a single event. This is an important limitation, as in real-life applications, sequential relationships can involve several events. Moreover, the algorithm of Das et al. [4] is highly inefficient as it tests all possible rules, without any strategy for pruning the search space.

The algorithm of Harms et al. [9] discovers sequential rules from sequence databases, and does not restrict the number of events contained in each rule. It searches for rules with a confidence and a support higher or equal to a user-specified threshold. The support of a rule is defined as the number of times that the right part occurs after the left part within user-defined time windows.

The algorithms of Das et al. [4] and Harms et al. [9] have been designed for mining rules occurring *frequently* in sequences. For this reason, these algorithms are inadequate for discovering rules *common* to many sequences. We illustrate this with an example. Consider a sequence database where each sequence corresponds to a customer and each event represents item(s) bought during a particular day. Suppose that one wishes to mine sequential rules that are common to many customers. The algorithms of Das et al. [4] and Harms et al. [9] are inappropriate for this task since a rule that appears many times in the same sequence could have a high support even if it does not appear in any other sequences.

Up to now, only Lo et al. [14], Pitman and Zanker [36] and Zhao et al. [34, 35] have proposed algorithms for extracting rules common to several sequences. Their algorithms discover rules such that their left and right parts are sequences of itemsets (sets of items). For instance, consider the rule {*Paganini*},{*Beethoven, Mozart*},{*Vivaldi*}⇒{*Berlioz*}, which means that customers who bought the music of Paganini, the music of Beethoven and Mozart *together*, and Vivaldi, have *then* bought the music of Berlioz. According to the definitions of Lo et al, Pitman and Zanker and Zhao et al., this rule is distinct from all its variations such as:

- {*Paganini*}, {*Beethoven*, *Vivaldi*}, {*Mozart*}⇒{Berlioz},
- {*Paganini, Beethoven, Vivaldi*}, {*Mozart*}⇒{Berlioz},
- {*Mozart*}, {*Paganini*}, {*Beethoven*}, {*Vivaldi*}⇒{*Berlioz*} and
- {*Vivaldi, Mozart, Paganini, Beethoven*}⇒{*Berlioz*}.

Therefore, if several of these rules appear in a database, the algorithms of Lo et al., Pitman and Zanker and Zhao et al. will fail to recognize that they describe a similar situation, which is that customers bought music from Paganini, Mozart, Beethoven and Vivaldi followed by music from Berlioz. This restrictive definition of what is a sequential rule can be an important problem because the measures that are used to determine if a rule is interesting (the support and the confidence) are based on the number of times that a rule and its left part appear in a database. As a consequence, the algorithms of Lo et al., Pitman and Zanker and Zhao et al. would not output any of the previous rules if each one taken individually does not meet the minimum interestingness criteria even if taken as a whole they represent an interesting pattern.

A second related concern with the algorithms of Lo et al., Pitman and Zanker and Zhao et al. is that because a rule and its variations may have important differences in their interestingness values, it can give a wrong impression of the relationships between items in the database to the user. For example, it is possible that the rule {*Paganini*}, {*Beethoven, Mozart*}, {*Vivaldi*} ⇒ {*Berlioz*} could have a high confidence, whereas one of its variation (e.g. {*Paganini*}, {*Beethoven, Mozart*}, {*Vivaldi*}⇒{*Berlioz*}) may have a low confidence.

A third concern is that because rules are very specific, they are less likely to match with a novel sequence of events

to make predictions. For example, consider the rule {*Mozart*}, {*Paganini*}, {*Beethoven*}, {*Vivaldi*} ⇒ {*Berlioz*}. This rule can be used to predict that a customer will buy the music of Berlioz if he bought Mozart, Paganini, Beethoven and Vivaldi, in that order. However, if the customer bought the items in a different order, the rule does not match and therefore cannot be used for making predictions.

For all of these reasons, we consider this form of rules to be too restrictive. As an alternative, we suggest discovering a more general form of sequential rules such that the order of the events in the left part and the right part are unordered. To mine these rules, we propose a new algorithm that we name CMRules [17].

CMRules in the contrary of the Manila et al. [13] and Harms et al.' algorithms [9] does not rely on a sliding-window approach. Instead, it is based on association rule mining. It first finds associations rules between items to prune the search space to items that occur jointly in many sequences. It then eliminates association rules that do not meet the minimum confidence and minimum support thresholds according to the time ordering. In this paper, we prove that this procedure discovers all sequential rules. The reason of using association rule mining is that for some datasets, many sequential rules are association rules (this is confirmed in our experiments).

The paper is organized as follows. The next section describes the problem of mining association rules, the problem of mining sequential rules common to several sequences, and analyzes the relation between sequential rules and association rules. The third section presents CMRules, proofs of correctness and completeness, and an analysis of its time complexity. The fourth section compares the performance of CMRules with an algorithm from the literature. The fifth section describes the application of CMRules in an intelligent tutoring agent that assists learners during learning activities. Finally, the last section draws a conclusion and discusses extensions and future work.

2 Background and Problem Definition

Association rule mining [1] is a popular knowledge discovery technique for discovering associations between items from a transaction database. Formally, a transaction database D is defined as a set of transactions $T=\{t_1,t_2\ldots t_n\}$ and a set of items $I=\{i_1, i_2, \ldots i_m\}$, where $t_1, t_2,\ldots, t_n \subseteq I$. The support of an itemset $X \subseteq I$ for a database is denoted as sup(X) and is calculated as the number of transactions that contains X. The problem of mining association rules from a transaction database is to find all association rules $X \rightarrow Y$, such that $X, Y \subseteq I$, $X \cap Y = \emptyset$, and that the rules respect some minimal interestingness criteria. The two interestingness criteria initially proposed [1] are that mined rules have a support greater or equal to a user-defined threshold *minsup* and a confidence greater or equal to a user-defined threshold *minconf*. The support of a rule $X \rightarrow Y$ is defined as $sup(X \cup Y) / |T|$. The confidence of a rule is defined as $conf(X \rightarrow Y) = sup(X \cup Y) / sup(X)$. The following property holds for any association rule:

Property 1: For any association rule $r: X \rightarrow Y$ in a transaction database T, the relation $conf(r) \geq sup(r)$ holds.
Proof. By definition, $sup(X \rightarrow Y) = sup(X \cup Y) / |T|$ and $seqConf(X \Rightarrow Y) = sup(X \cup Y) / sup(X)$. Because, $|T| \geq sup(X)$, the property is true.

Association rules are mined from transaction databases. A generalization of a transaction database that contains time information about the occurrence of items is a sequence database [2]. A sequence database SD is defined as a set of sequences $S=\{s_1, s_2\ldots s_n\}$ and a set of items $I=\{i_1, i_2,\ldots i_n\}$, where each sequence sx is an ordered list of transactions (itemsets) $sx=\{X_1, X_2, \ldots X_m\}$ such that $X_1, X_2, \ldots X_n \subseteq I$.

We propose the following definition of a sequential rule to be discovered in a sequence database. A sequential rule $X \Rightarrow Y$ is a relationship between two itemsets X,Y such that $X,Y \subseteq I$ and $X \cap Y = \emptyset$. The interpretation of a rule $X \Rightarrow Y$ is that if the items of X occur in some transactions of a sequence, the items in Y will occur in some transactions afterward from the same sequence. Note that there is no ordering restriction between items in X and between items in Y (they are not required to occur in the same itemset of a sequence). To evaluate the interestingness of sequential rules, we define two measures, which are an adaptation for multiple sequences of the measures used by other sequential rule mining algorithms [4, 9, 13]. The first measure is the rule's sequential support and is defined as $seqSup(X \Rightarrow Y) = sup(X \blacksquare Y) / |S|$. The second measure is the rule's sequential confidence and is defined as $seqConf(X \Rightarrow Y) = sup(X \blacksquare Y) / sup(X)$. The notation $sup(X \blacksquare Y)$ denotes the number of sequences from a sequence database where all the items of X appear before all the items of Y (note that items from X or from Y do not need to be in the same transaction). The notation sup(X) denotes the number of sequences that contains X. The following property holds for any sequential rule.

Property 2: For any sequential rule $r: X \Rightarrow Y$ in a database S, the relation $seqConf(r) \geq seqSup(r)$ holds.
Proof. By definition, $seqSup(X \Rightarrow Y) = sup(X \blacksquare Y) / |S|$ and $seqConf(X \Rightarrow Y) = sup(X \blacksquare Y) / sup(X)$. Because, $|S| \geq sup(X)$, the property is true.

We define the *problem of mining sequential rules common to multiple sequences* as the problem of finding all valid rules in a sequence database. A *valid rule* is a rule having a sequential support and a sequential confidence respectively no less than user-defined thresholds *minSeqSup* and *minSeqConf*. As an example, consider the sequence database shown in the left part of figure 1. The right part of figure 1 illustrates some valid rules found in this database for *minSeqSup* = 0.5 and *minSeqConf* = 0.5. Consider the rule {a, b, c} ⇒ {e}. Its sequential support is sup({a, b, c}■{e}) / |S| = 2 / 4 = 0.5 and its sequential confidence is sup({a, b, c}■{e}) / sup(X) = 2 / 2 = 1. Because those values are no less than *minSeqSup* and *minSeqConf*, the rule is valid.

Note that the problem of mining sequential rules as defined here is largely different from the one of mining sequential patterns, since items from the left and the right part of a sequential rule are unordered. Therefore it is not possible to simply adapt a sequential pattern mining algorithm to generate these rules.

The CMRules algorithm that we propose in this paper is based on the observation that if we ignore or remove the time information of a sequence database SD, we obtain a transaction database SD'. For each sequence database SD and its corresponding transaction database SD', each sequential rule r: X⇒Y of S has a corresponding association rule r': X→Y in S'. Two properties are proven for all such rule r and r'.

Property 3: For any sequential rule r: X⇒Y in a sequence database SD, the relation sup(r') ≥ seqSup(r) holds.

Proof. By definition the sequential support of r and the support of r' are respectively sup(X■Y) / |SD| and sup(X∪Y) / |SD'|. Because |SD| = |SD'| and sup(X∪Y) ≥ sup(X■Y) the relation sup(r') ≥ seqSup(r) holds.

Property 4: For any sequential rule r: X⇒Y in a sequence database SD, the relation conf(r') ≥ seqConf(r) holds. **Proof.** By definition the sequential confidence of r and the confidence of r' are respectively sup(X■Y) / sup(X) and sup(X∪Y) / sup(X). Because sup(X∪Y) ≥ sup(X■Y), the relation conf(r') ≥ seqConf(r) holds.

ID	Sequences
1	{a, b},{c},{f},{g},{e}
2	{a, d},{c},{b},{ e, f}
3	{a},{b},{f},{e}
4	{b},{f, g}

→

ID	Rule	S-Support	S-Confidence
r1	{a, b, c} ⇒ {e}	0.5	1.0
r2	{a} ⇒ {c, e, f}	0.5	0.66
r3	{a, b} ⇒ {e, f}	0.5	1.0
r4	{b} ⇒ {e, f}	0.75	0.75
r5	{a} ⇒ {e, f}	0.75	1.0
r6	{c} ⇒ {f}	0.5	1.0
r7	{a} ⇒ {b}	0.5	0.66
…	…	…	…

Figure 1: A sequence database (left) and sequential rules found (right).

3 THE CMRULES ALGORITHM FOR MINING SEQUENTIAL RULES

Based on the previous observations of the relationship between sequential rules and association rules, we propose the CMRules algorithm for mining sequential rules. The algorithm is presented in figure 2. Figure 3 shows a sample execution with a sequence database containing four sequences.

The algorithm takes as input a sequence database and the *minSeqSup* and *minSeqConf* thresholds and outputs the set of all sequential rules in the database respecting these thresholds. CMRules starts by ignoring the sequential information from the sequence database to obtain a transaction database (fig. 2 and 3. Step 1). It then applies an association rule mining algorithm to discover all association rules from this transaction database with *minsup* = *minSeqSup* and *minconf* = *minSeqConf* (fig. 2 and 3. Step 2). The algorithm then calculates the sequential support and sequential confidence of each association rule by scanning the sequence database, and then it eliminates the rules that do not meet the *minSeqSup* and *minSeqConf* minimum thresholds (fig. 2 and 3. Step 3). The set of rules that is kept is the set of all sequential rules.

INPUT: a sequence database, ***minSeqSup, minSeqConf***
OUTPUT: the set of all sequential rules
PROCEDURE:

1. Consider the sequence database as a transaction database.
2. Find all association rules from the transaction database by applying an association rule mining algorithm such as Apriori [1]. Select *minsup* = *minSeqSup* and *minconf* = *minSeqConf*.
3. Scan the original sequence database to calculate the sequential support and sequential confidence of each association rule found in the previous step. Eliminate each rule *r* such that:
   a. seqSup(r) < *minSeqSup*
   b. seqConf(r) < *minSeqConf*
4. Return the set of rules.

Figure 2: The CMRules algorithm

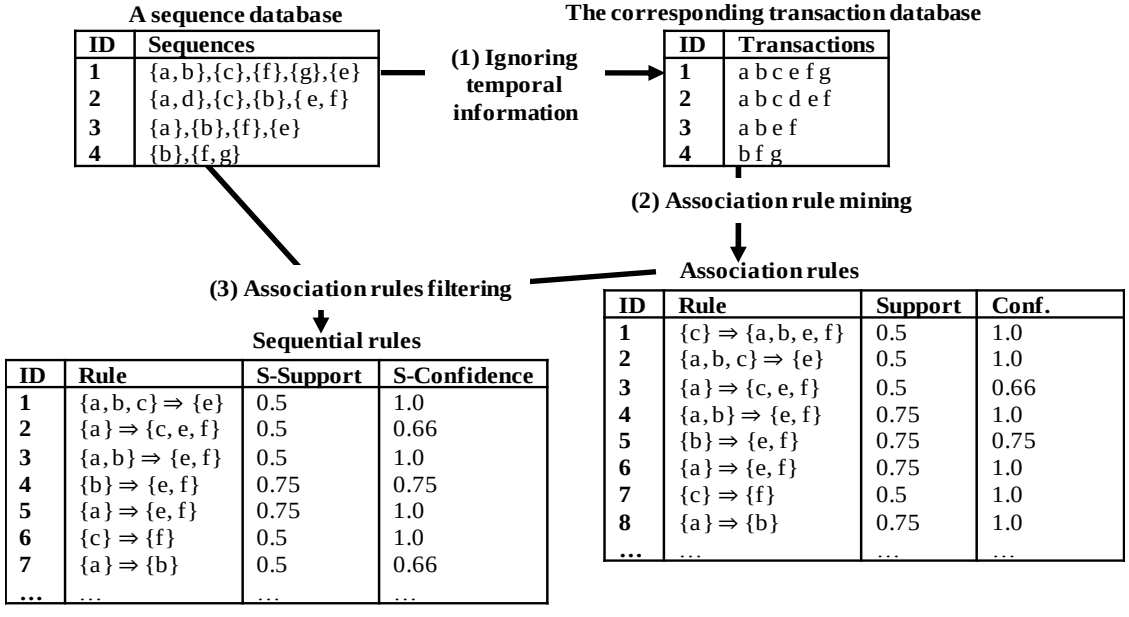

Figure 3: A Sample execution of the CMRules algorithm

3.1 Proof of Completeness

To prove that CMRules can find all sequential rules, we have to demonstrate that the set of all sequential rules is a subset of the set of all association rules. This is proven next.

Theorem 1. The algorithm discovers all sequential rules for a given *minSeqConf* and *minSeqSup,* if we choose *minconf* ≤ *minSeqConf* and *minsup* ≤ *minSeqSup*. **Proof**. To prove this, we can consider the sequential support and the sequential confidence separately, without loss of generality.

Let's first consider the sequential support. Suppose there is a sequential rule r such that seqSup(r) ≥ *minSeqSup* and *minsup* > sup(r′) (we suppose that the corresponding association rule r′ is not frequent). Property 3 states that sup(r′) ≥ seqSup(r). Therefore, it follows that *minsup* > sup(r′) ≥ seqSup(r) ≥ *minSeqSup*. Thus, *minsup* > *minSeqSup*. Therefore, if we choose *minSeqSup* ≥ *minsup* there will be no such rule r.

Now consider the sequential confidence. Suppose there is a sequential rule r such that seqConf(r) ≥ *minSeqConf* and that *minconf* > conf(r′). Property 4 states that conf(r′) ≥ seqConf(r). Therefore *minconf* > conf(r′) > seqConf(r) ≥ *minSe-*

qConf. Thus, *minconf* > *minSeqConf*. Therefore, if we choose *minSeqConf* ≥ *minconf* there will be no such rule r.

Corollary 1. The algorithm is more effective if we choose *minconf* = *minSeqConf* and *minsup* = *minSeqSup*. **Proof**: In association rules mining, the lower *minsup* or *minconf* is, the more rules can be found, and the more computation can be required. Therefore, for our algorithm it is best to select the highest possible value for *minsup* and *minconf* that will discover association rules containing all desired sequential rules. In Theorem 1, we found that these values are bound by *minSeqConf* and *minSeqSup*, to have a guarantee that all sequential rules will be found. Therefore, the highest value that one can give to *minconf* and *minsup* are respectively *minSeqConf* and *minSeqSup*.

3.2 Proof of Correctness

We now demonstrate that CMRules is a correct algorithm. To do this, we have to show that it does not generate invalid sequential rules.

Theorem 2. The CMRules algorithm (fig.2) does not generate invalid rules. **Proof.** We know from Theorem 1 that if we choose *minconf* ≤ *minSeqConf* and *minsup* ≤ *minSeqSup* for association rule mining, the resulting set of association rules contains all valid sequential rules respecting *minSeqSup* and *minSeqConf*. But because the relations sup(r′) ≥ seqSup(r) and conf(r′) ≥ seqconf(r) hold for any rule sequential rule r (cf. Property 3 and Property 4), the set of association rules can also contains invalid sequential rules. However, the CMRules does not output invalid rules because Step 3 of the algorithm will filter them out. Therefore, the algorithm is correct.

3.3 Implementing Step 3 Efficiently

We now describe how to implement Step 3 of the CMRules algorithm efficiently. The naive approach would be to take each sequence and check if each association rule X→Y is contained in it to calculate the sup(X■Y) value that is necessary for calculating the sequential confidence and sequential support. Checking if a rule is included in a sequence is done in linear time (a sequence can be scanned one time from left to right to see if X occurs, and Y occurs afterward). However, performing this check for all sequences of a sequence database is inefficient because not all sequences contain the itemsets of each rule. We describe next how to minimize the number of sequences to be checked for each rule. But we first need to explain how association rule mining is performed.

To mine association rules, one needs to apply an algorithm such as Apriori [1]. Algorithms for association rule mining proceed in two steps[1]. They first discover frequent itemsets, and then use them to generate association rules. A frequent itemset is an itemset that appear more than *minsup* times in a transaction database. The support of an itemset is defined as the number of transactions that contains it. Generating association rules is then done by selecting pairs of frequent itemsets X and Y such that X⊆ Y, to generate rules of the form X→Y-X (see [1] for a fast algorithm).

Now, to implement efficiently step 3 of CMRules, we need to modify the frequent itemset mining algorithm so that each itemset X found is annotated with the set of transactions that contains it. This is a trivial modification for an algorithm such as Apriori [1] (in the case of Apriori, this variation is known as Apriori-TID [1]). Having this extra information about frequent itemsets, Step 3 can be performed efficiently by checking each rule X→Y only against the sequences that contain its left itemset X. This will allow calculating correctly the sequential support and sequential confidence of the rule, since sup(X■Y) can be calculated by checking only with the sequences containing X (by the definition of sup(X■Y)), and the other terms required for calculating the sequential support and confidence are known (sup(X) and |S|). In our tests, this simple optimization improved the performance of Step 3 by up to 50%.

3.4 Reducing Memory Consumption

To reduce memory consumption, it is possible to merge Step 2 and Step 3 of CMRules so that each association rule is immediately checked for its sequential support and confidence. This allows having only one association rule at a time in central memory. Therefore, there is no need to store the set of all association rules. This greatly reduces memory consumption and also makes the algorithm faster. Moreover, sequential rules can be saved to disk immediately after being found, and if one wish to keep association rules, these latter can be saved to the disk at the same time. This implementation strategy greatly reduces the memory requirement of the algorithm.

3.5 Analysis of the Time complexity

We here provide a brief analysis of the time complexity of CMRules. First, the time complexity of converting a sequence database in a transaction database (Step 1) is linear with respect to the number of sequences and their sizes.

The time complexity of Step 2 is more difficult to establish. It depends on which association rule mining algorithm is used. As explained previously, mining association rule is done in two phases: mining frequent itemsets and generating rules. The first phase is the most costly [1], so it is adequate to ignore the second phase for estimating the time complexity. If we use the Apriori algorithm for mining frequent itemsets, the time complexity is $O(d^2n)$ where d is the number of different items and n is the number of transactions in the database (see [10] for a proof). In our implementation, we used Apriori-TID a variation that performs slightly better for some datasets [1] and has a similar time com-

plexity [10].

Step 3 of CMRules checks each candidate rule against the set of sequences that contains its antecedent (as explained). In the best case and worst case, there are respectively |S| × *minsup* sequences, and |S| sequences to be considered for each rule. Checking if a rule appear in a sequence is done in linear time. Thus, the time complexity of step 3 is linear with respect to the number of sequences that contain the antecedent of each rule, the size of each sequence and the total number of rules.

4 Evaluation of the Algorithm

We have implemented CMRules in Java. We next compare its performance with an adaptation of the Deogun et al. [5] algorithm on real-life datasets. We chose the Deogun et al. algorithm as baseline algorithm for evaluating CMRules because it can be adapted to produce the same result as CMRules. We did not compare with other algorithms such as the algorithm of Lo et al., Zhao et al. and Pitman and Zanker because they do not produce the same results and use very different process for finding rules. Experiments were performed on a notebook computer with a 2.53 Ghz P8700 Core 2 Duo processor running Windows XP and 1 GB of free RAM.

4.1 The CMDeo algorithm

Although the Deogun et al. algorithm was proposed for mining sequential rules in a single sequence of events, it is easy to adapt it for mining rules common to multiple sequences, as defined in this article. We name this adaptation CMDeo and have implemented it in Java. Since this is not the main subject of this article, we will briefly describe the main idea of this algorithm. Before this explanation, we give two definitions. We say that the *size* of a sequential rule is k*n, if the left part and right part contain respectively k and n items. For example, the rules {a, b, c}⇒{e, f, g} and {a}⇒{e, f} are of size 3*3 and 1*2, respectively. Furthermore, a rule of size f*g is said to be *larger* than another rule of size h*i if f > h and g ≥ i, or alternatively if f ≥ h and g > i.

The CMDeo algorithm works as follows. It first finds all valid rules of size 1*1. This is done in three steps. First, the CMDeo counts the support of single items by scanning the sequence database once. Then, for each pair of frequent items x,y appearing in at least *minSeqSup* same sequences, the algorithm generates candidates rules {x}⇒{y} and {y}⇒{x}. Then, the sequential support and sequential confidence of each rule is calculated as in Step 3 of CMRules. Each rule respecting *minSeqSup* and *minSeqConf* is noted as being a valid rule of size 1*1.

Then, CMDeo discovers larger valid rules in a level-wise manner (similar to Apriori [1]) by recursively applying two processes: a) *left expansion,* and b) *right expansion*. Their role is to combine pair of rules with the same size to obtain larger candidate rules.

Left-side expansion is the process of taking two rules X⇒Y and Z⇒Y, where X and Z are itemsets of size n sharing n-1 items, to generate a larger rule$X \cup Z \Rightarrow Y$. Right-side expansion is the process of taking two rules Y⇒X and Y⇒Z, where X and Z are itemsets of size n sharing n-1 items, to generate a larger rule $Y \Rightarrow X \cup Z$. These two processes can be applied recursively to find all rules starting from rule of size 1*1 (for example, rules of size 1*1 allows finding rules of size 1*2 and rule of size 2*1). Two properties are proven for left and right expansions regarding their effect on sequential support:

Property 5 (left expansion, effect on sequential support): If an item i is added to the left side of a rule r:X⇒Y, the sequential support of the resulting rule r° is lower or equal to the sequential support of r. **Proof**: The sequential support of r and r° are respectively $sup(X \blacksquare Y)/|S|$ and $sup(X \cup \{i\} \blacksquare Y)/|S|$. Since $sup(X \blacksquare Y) \geq sup(X \cup \{i\} \blacksquare Y)$, $seqSup(r) \geq seqSup(r°)$.

Property 6 (right expansion, effect on sequential support): If an item i is added to the right side of a rule r:X⇒Y, the sequential support of the resulting rule r° can be lower or equal to the sequential support of r. **Proof**: The sequential support of r and r° are respectively $sup(X \blacksquare Y)/|S|$ and $sup(X \blacksquare Y \cup \{i\})/|S|$. Since $sup(X \blacksquare Y) \geq sup(X \blacksquare Y \cup \{i\})$, $seqSup(r) \geq seqSup(r°)$.

The two previous properties imply that the sequential support is monotonic with respect to left and right expansions. In other words, performing left/right expansions will always results in rules having a sequential support that is lower or equal to the sequential support of the original rules. Therefore, all rules having a sequential support higher or equal to *minSeqSup* can be found by recursively performing expansions starting from rules of size 1*1 that have a sequential support higher or equal to *minSeqSup*. Moreover, properties 5 and 6 guarantee that expanding a rule having a sequential support less than *minSeqSup* will not result in a rule having a sequential support higher or equal to *minSeqSup*. This is very important because it allows pruning the search space. For the sequential confidence, however, the situation is different as it is explained in the next two next properties.

Property 7 (left expansion, effect on sequential confidence): If an item i is added to the left side of a rule r:X⇒Y, the sequential confidence of the resulting rule r° can be lower, higher or equal to the sequential confidence of r. **Proof**: The sequential confidence of r and r° are respectively sup(X■Y)/|sup(x)| and sup(X∪{i}■Y)/sup(X∪{i}). Because sup(X■Y) ≥ sup(X∪{i}■Y) and sup(X) ≥ sup(X∪{i}), seqConf(r) can be lower, higher or equal to seqConf(r°).

Property 8 (right expansion, effect on sequential confidence): If an item i is added to the right side of a rule r:X⇒Y, the sequential confidence of the resulting rule r° can be lower or equal to the sequential confidence of r. **Proof**: The sequential confidence of r and r° are respectively sup(X■Y)/|sup(x)| and sup(X■Y∪{i})/sup(X). Since sup(X■Y) ≥ sup(X■Y∪{i}), seqConf(r) ≥ seqConf(r°).

Because of property 7, the sequential confidence is not monotonic with respect to left and right expansions and therefore it cannot be used to prune the search space. Therefore, CMDeo only use the sequential support to prune the search space.

CMDeo proceeds as follows. First, it finds all valid rules of size 1*1, as we have explained. Then it takes all pairs of rule of size 1*1 having the minimal support and it performs left/right expansions to generate candidate rules of size 2*1 and 1*2. For each candidate rule that is generated, CMDeo calculates the sequential support and sequential confidence by scanning the sequence(s) containing the left and right itemsets of the rule. If a rule is valid, CMDeo noted it as being valid. Then CMDeo recursively performs expansions with candidates of size 2*1 and 1*2 to find candidate rules of size 3*1, 2*2 and 1*3. Then CMDeo evaluates these rules to see if they are valid, and this process continues recursively until no pair or rules having the minimum support can be expanded.

This is the basic principle by which CMDeo finds all valid rules. However, two further problems have to be solved to ensure that CMDeo does not find some rules twice.

The first problem is that some rules can be found by different combinations of left/right expansions. For example, consider the rule {a, b}⇒{c, d}. By performing, a left and then a right expansion of {a}⇒{c}, one can obtain the rule {a, b}⇒{c, d}. However, the aforementioned rule can also be obtained by performing a right and then a left expansion of the rule {a}⇒{c}. This problem is illustrated in Figure 4(A). For example, rules of size 2*2 can be found respectively by expanding the left part of rules of size 1*2 and by expanding the right part of rules of size 2*1. A simple solution to this problem is to not allow performing a right expansion after a left expansion. By applying this solution, rules will be discovered according to the order presented on Figure 4(B). Note that an alternative solution would be to not allow a left expansion after a right expansion.

The second problem is that rules can be found several times by performing left/right expansions with different items. For example, consider the rule {b, c}⇒{d}. A left expansion of {b}⇒{d} with item c results in the rule {b, c}⇒{d}. But the latter rule can also be found by performing a left expansion of {c}⇒{d} with b. To solve this problem, CMDeo only add an item to a rule's itemset if the item is greater than all items in the itemset according to the lexicographic ordering. In the previous example, this means that item c will be added to the left side of {b}⇒{d} (because c is greater than b). However, item b will not be added to the left side of {c}⇒{d} (because b is smaller than c). By using this strategy and the previous one, no rule is found twice.

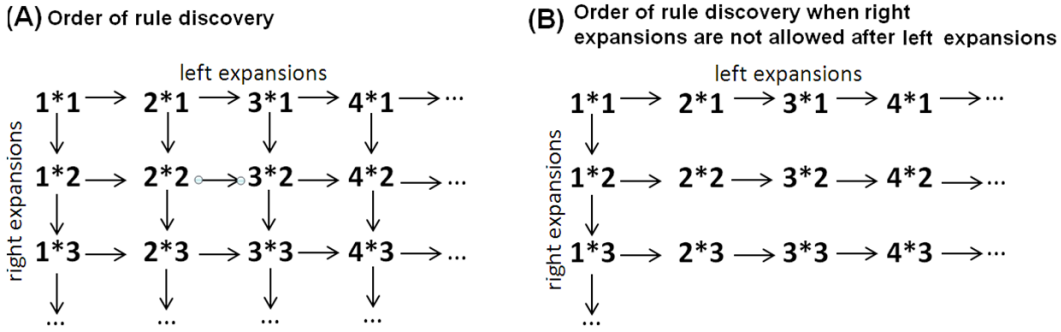

Figure 4: How rules are discovered by left/right expansions with CMDeo

It is important to note that the original algorithm of Deogun et al. is designed to find a subset of all valid sequential rules. Therefore, there is no guarantee that it finds all the valid rules. In this paper, we have adapted the algorithm of Deogun et al. so that it finds all valid sequential rules. Moreover, we have adapted the algorithm of Deogun et al. for the case of multiple sequences (the original algorithm mines rules from a single sequence). Our implementations of CMRules and CMDeo can be downloaded from http://www.philippe-fournier-viger.com/spmf/.

Table 1. Datasets characteristics

	Kosarak-1	*Kosarak-2*	*BMS-Webview1*	*Toxin-Snake*
Number of sequences	70,000	41,582	59,601	163
Number of items	21,144	25,257	497	20
Number of items by itemset	1	3	1	1
Average number of items by sequence	7.97 (σ =21.14)	26.51 (σ = 28.25)	2.51 (σ =4.85)	60.61 (σ =0.598)
Average number of different items by sequence	7.97 (σ =21.14)	26.51 (σ =28.25)	2.51 (σ =4.85)	17.84 (σ =1.09)

4.2 Description of the datasets

To compare the performance of CMRules and CMDeo, we have performed a series of experiments with real datasets used in the data mining literature. The characteristics of these datasets are presented in table 1.

The two first datasets are derived from Kosarak, a public dataset available from http://fimi.cs.helsinki.fi/data/. This latter contains 990,000 click-stream data from the logs of an online news portal. **Kosarak-1** is a smaller version of Kosarak that contains only the 70,000 first sequences of Kosarak. We have kept only the first 70,000 sequences to make the experiment faster. These sequences have 7.97 items on average. **Kosarak-2** is a second variation of the Kosarak dataset that we have created to test the algorithms on longer sequences. In Kosarak-2, only sequences from Kosarak that have more than 9 items are used. This results in 41,582 long sequences with an average of 26.51 items by sequence. Moreover, because Kosarak only contains 1 item by itemset, in Kosarak-2, it was decided to group consecutive items three by three into itemsets.

The third dataset is **BMS-Webview1**, a sequence database containing several months of click-stream from an e-commerce website. It was used for the KDD-Cup 2001 competition and can be downloaded from http://www.ecn.purdue.edu/KDDCUP/. BMS-Webview1 differs from Kosarak-1 and Kosarak-2 principally in that it contains shorter sequences and the set of different items is much smaller than those of Kosarak1 and Kosarak2.

The fourth dataset is **Toxin-Snake** [19], a sequence database from the domain of biology. It contains 192 protein sequences. For our experiments, only sequences containing more than 50 items were kept. Keeping only these sequences has been done to make the dataset more uniform (because the original Snake dataset contains a few sequences that are very short and many long sequences). This resulted in 163 long sequences containing an average of 60.61 items. Besides having longer sequences than those from the three other datasets, Toxin-Snake is a very dense dataset. Each item occurs in almost every sequence (there is on average 17.84 different items in each sequence and only 20 different items for the whole dataset) and each item appearing in a sequence appears on average 3.39 times in the sequence (σ =2.24).

4.3 Experiment to assess the influence of *minSeqSup*

The first experiment consists of running CMRules and CMDeo on the four datasets with different values for *minSeqSup* and a fixed value for *minSeqConf*, to assess the influence of *minSeqSup* on the performance of CMRules and CMDeo.

For **Kosarak-1**, CMRules and CMDeo were run with with *minSeqConf* = 0.3 and *minSeqSup* = 0.004, 0.00375… 0.002. For this experiment, a time limit of 1200 seconds and a memory limit of one gigabyte of memory were used. Figure 5 shows the execution times and maximum memory usage of each algorithm for each *minSeqSup* value, as well as the number of sequential rules and association rules found. Note that no result are shown for CMDeo for *minSeqSup* < 0.0025 because CMDeo exceeds the memory limit, and no result are shown for CMRules for *minSeqSup* < *0.002* because CMRules exceeds the time limit. During this experiment, CMRules remained on average two times faster than CMDeo for all *minSeqSup* values and used up to five times less memory. The number of sequential rules found ranged from five to eleven percent of all association rules. This shows that the percentage of sequential rules can be quite high in real datasets. We also observed that the execution time of CMRules seems to grow proportionally to the number of association rules. For the lowest *minSeqSup* value, valid sequential rules contained on average 3.72 items with a maximum of six.

For **Kosarak-2,** we applied both algorithms with *minSeqConf* = 0.3 and *minSeqSup* = 0.15, 0.145… 0.065. We set a memory limit of one gigabyte and a time limit of 1200 seconds. Results of this experiments are shown on figure 6. For high *minSeqSup* values, CMDeo was slightly faster than CMRules, but as *minSeqSup* got lower, CMRules became up to two times faster. The maximum memory usage of each algorithm was similar. For this dataset, results are not shown for *minSeqSup* < 0.065 because both algorithms exceeded the time limit. The percentage of association rules that were sequential rules varied between nine and fourteen percent. Sequential rules contained 3.62 items on average, with a maximum of seven items. As for previous datasets, it was observed that the performance of CMRules grows proportionally to the number of association rules.

For **BMS-Webview1,** we applied both algorithms with *minSeqConf* = 0.3 and *minSeqSup* = 0.00085, 0.000825, …, 0.00065. Figure 7 illustrates the results. A time limit of 1000 seconds and a memory limit of 1GB were used. For this reason, no result is shown for CMRules and CMDeo for *minSeqSup* < 0.000725 and *minSeqSup* < 0.00065, respectively. Overall, CMDeo was up to 3.87 times faster than CMRules. The percentage of association rules that were sequential

rules varied from eleven to seventeen percent of all association rules. By looking at figure 7, it is again clear that CMRules execution time varied proportionally to the number of association rules. For *minSeqSup* = 0.00065, sequential rules contained on average 5.25 items with a maximum of eleven items.

For **Toxin-Snake**, we applied both algorithms with *minSeqConf* = 0.3 and *minSeqSup* = 0.985, 0.98…, 0.93. Figure 8 illustrates the results. A time limit of 400 seconds and a memory limit of 1GB were set. For this reason, no result are shown for CMRules for *minSeqSup* < 0.93. Furthermore, we did not run CMDeo for values lower than *minSeqSup* = 0.93 because the trend clearly indicates that CMDeo performs better and has a better scalability than CMRules in terms of execution time for this dataset. The performance gap between the algorithms was the highest among all the experiments, CMDeo being 27 times faster than CMRules for *minSeqSup* = 0.93. In terms of memory usage, CMDeo performed slightly better than CMRules for this dataset. The percentage of association rules that are sequential rules varied from two to eleven percent. Sequential rules contained 5.06 items on average with a maximum of ten items.

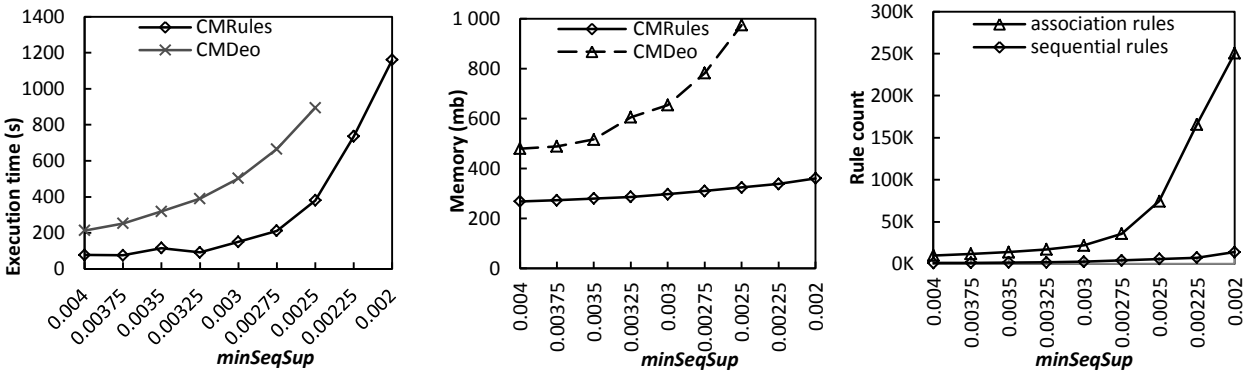

Figure 5. Influence of *minSeqSup* for the Kosarak-1 dataset

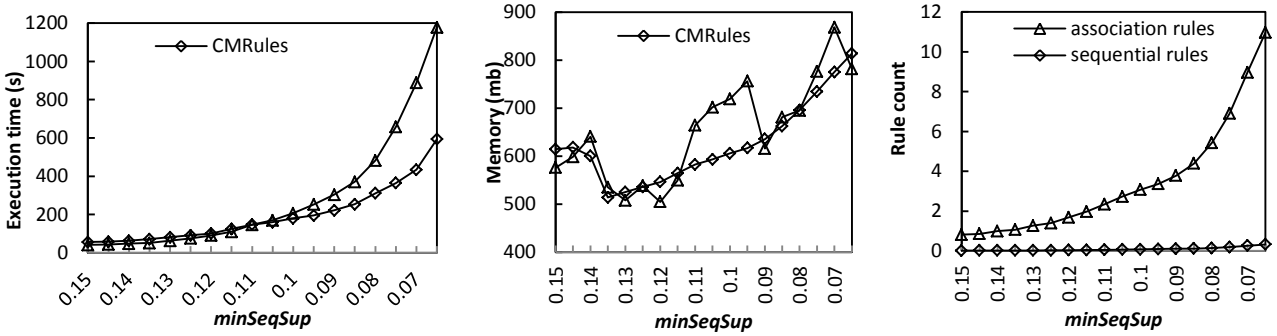

Figure 6. Influence of *minSeqSup* for the Kosarak-2 dataset

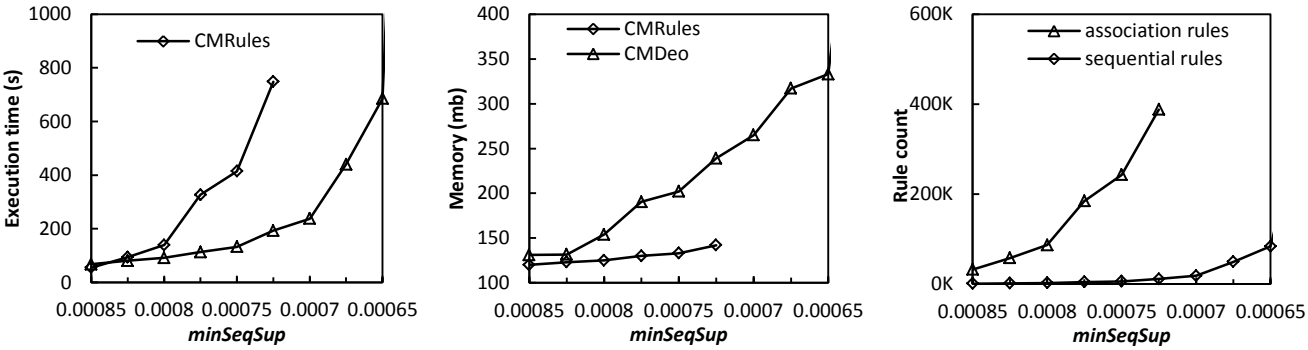

Figure 7. Influence of *minSeqSup* for the BMS-WebView1 dataset

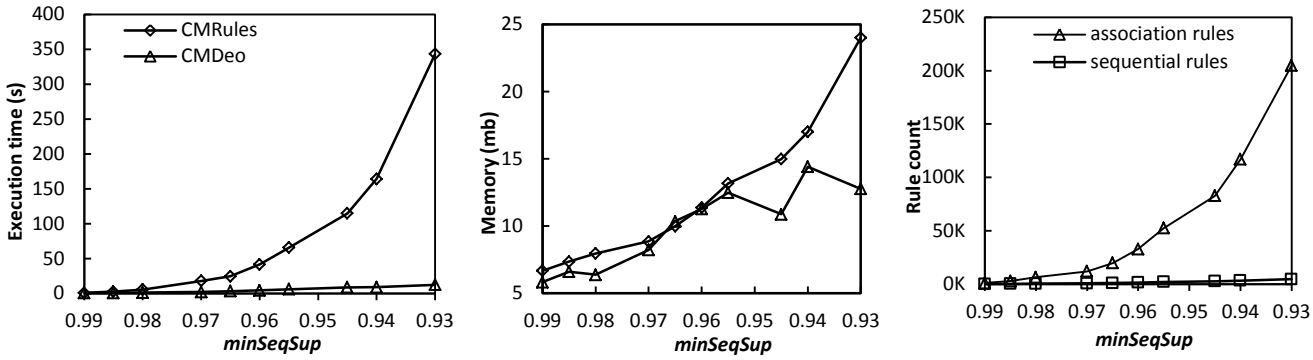

Figure 8. Influence of *minSeqSup* for the Toxin-Snake dataset

4.4 Experiment to assess the influence of *minSeqConf*

The second experiment compare the performance of CMDeo and CMRules for different *minSeqConf* values given the same four datasets. For this experiment, *minSeqSup* was set to the same values as in the previous experiment and *minSeqConf* was set to 0.3 and 0.8.

Figures 9, 10, 11 and 12 show execution times, numbers of association rules and number of sequential rules. The maximum memory use is not shown for this experiment because *minSeqConf* did not produce any significant difference on it. From the figures, it can be observed that the execution time of CMDeo did not vary much between *minSeqConf* = 0,3 and *minSeqConf* = 0,8. The reason is that CMDeo does not use the confidence to prune the search space. On the other hand, CMRules benefited from a higher *minSeqConf* threshold because it uses the confidence to prune the search space (CMRules can prune the search space based on the confidence because it is based on association rule mining). The execution time of CMRules decreased by up to 42%, 46 % and 55 % respectively for Kosarak-1, Kosarak-2 and BMS-WebView1. For Toxin-Snake however, *minSeqConf* did not had a big influence on the execution time of CMRules because it is a very dense dataset. Thus, extracted association rules and sequential rules have a very high sequential confidence (for example, for *minSeqSup* = 0.93, valid sequential rules have on average a confidence of 98 %).

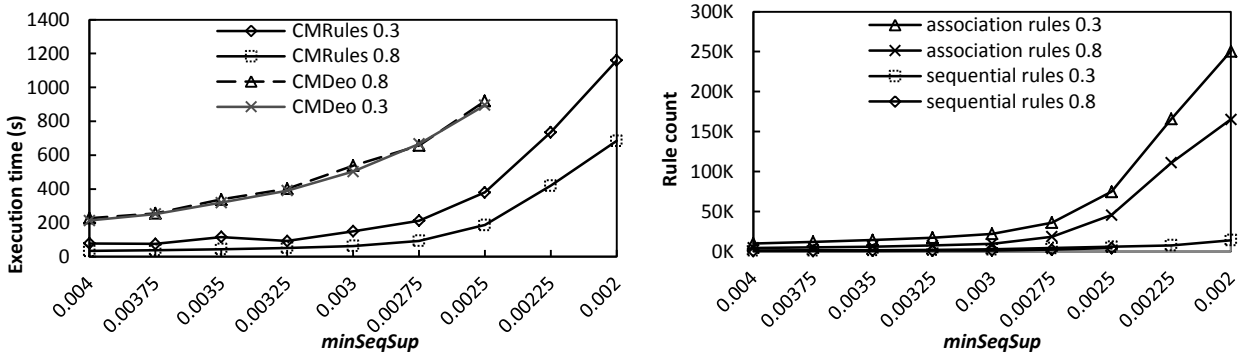

Figure 9. Influence of *minSeqConf* on execution time and rule count for the Kosarak-1 dataset

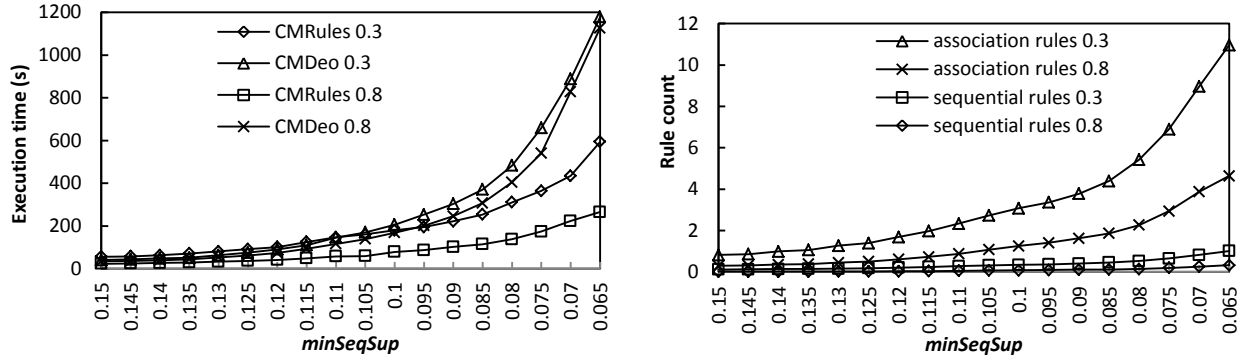

Figure 10. Influence of *minSeqConf* on execution time and rule count for the Kosarak-2 dataset

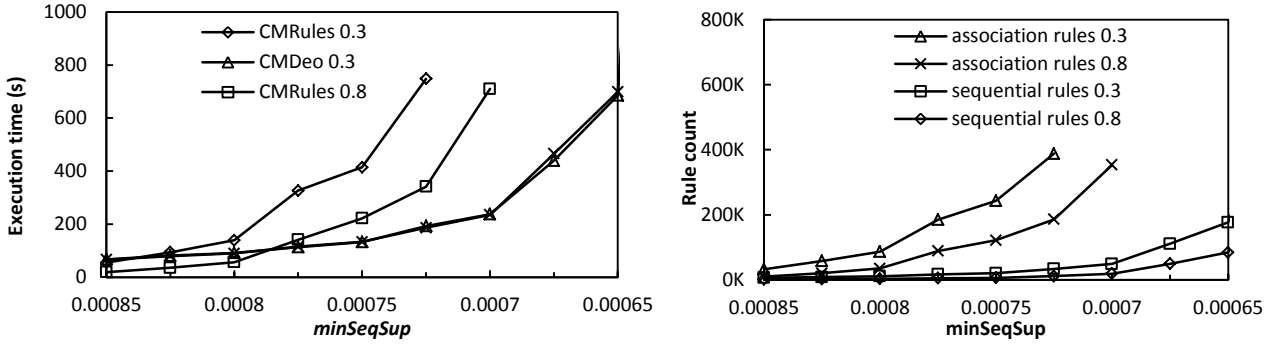

Figure 11. Influence of *minSeqConf* on execution time and rule count for the BMS-WebView1 dataset

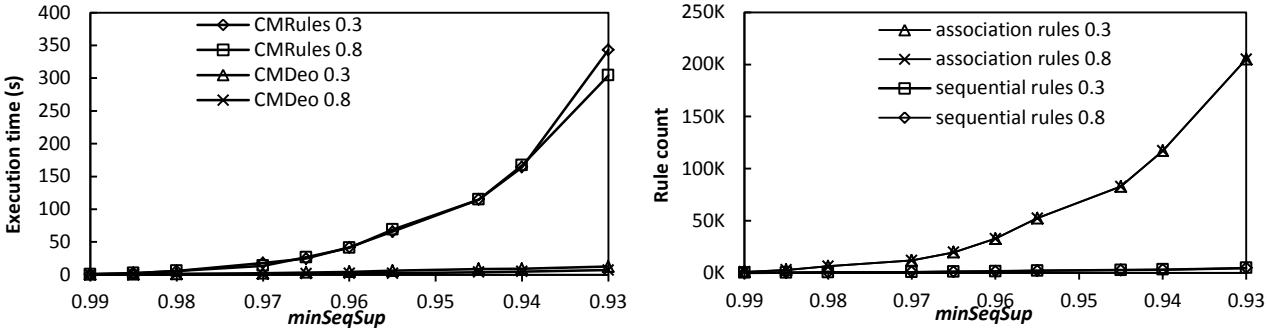

Figure 12. Influence of *minSeqConf* on execution time and rule count for the Toxin-Snake dataset

4.5 Experiment to assess the influence of |S|

A third experiment was performed to assess the scalability of CMRules and CMDeo with respect to |S|- the number of sequences contained in the sequence database. For this experiment, the original Kosarak dataset was used because it is a very large real-life dataset containing 990,000 sequences, which is convenient for varying the size of the dataset. Note that we also tried this experiment with Toxin-Snake and BMS-WebView1. But because Toxin-Snake is very small (only 163 sequences) and because we need to use a very low *minSeqSup* threshold for BMS-WebView1 to discover sequential rules (in the range of 0.0008 and 0.00065), the number of rules found and execution times varied too much with |S|, which made the results impossible to use for analyzing the scalability of the algorithms. For this reason, we only report the results for Kosarak.

For this experiment, CMRules and CMDeo were run with *minSeqSup* = 0.003 and *minSeqConf* = 0.5, while |S| was varied from 10,000 to 200,000 sequences with an increment of 10,000 sequences. The values for *minSeqSup* and *minSeqConf* were chosen so that both algorithms could be run until at least 100,000 sequences under a one gigabyte memory limit.

A time limit of 1,000 seconds was set. Figure 13 shows the results of the experiment (execution times, maximum memory usage and number of rules found). As it can be seen, CMRules's execution time and maximum memory usage grows linearly with the size of |S|. CMDeo shown a similar trend. However, as it approached the memory limit of one gigabyte, its performance was negatively affected by the Java garbage collection mechanism. No results are shown for |S|>100,000 and |S|>200,000 for CMDeo and CMRules respectively, because of the memory limit.

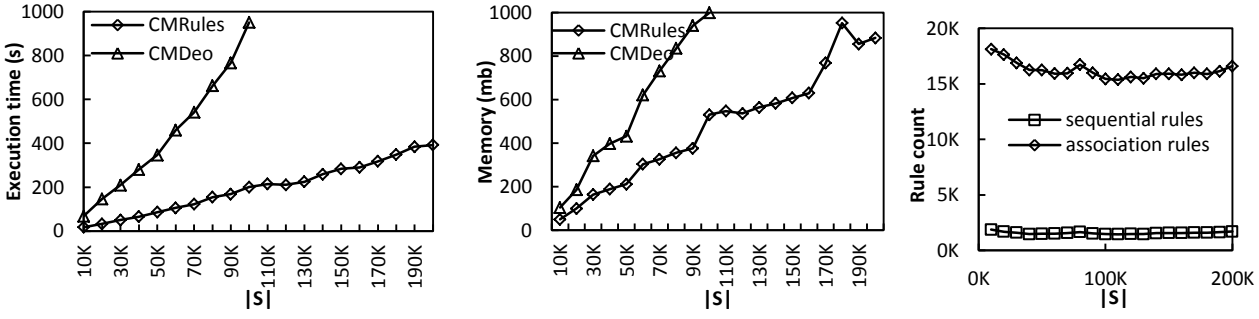

Figure 13. Result of the scalability experiment with Kosarak

4.5 Discussion of the results

In previous experiments, CMRules and CMDeo were both faster for certain datasets. CMRules performed better than CMDeo for Kosarak-1 and Kosarak-2, and CMDeo outperformed CMRules for BMS-Webview1 and Toxin-Snake.

Table2. Analysis of the results

	Kosarak-1	***Kosarak-2***	***BMS-Webview1***	***Toxin-Snake***
percentage of association rules that are sequential rules (CMRules)	5% to 11%	9 to 14 %	11 to 17 %	2 to 11 %
percentage of expansions resulting in valid rules (CMDeo)	2.7 %	1.44 %	10.05 %	43.56 %
average sequential rule support	0.37 %	10.51 %	0.08 %	95 %
average sequential rule confidence	75 %	68 %	73.8 %	98 %

To explain why CMRules and CMDeo performed better on some datasets, we took additional measurements. These measurements are shown in Table 2. The first row indicates the percentage of association rules that were sequential rules. The second row indicates the percentage of rule expansions in CMDeo that resulted in valid rules for the lowest *minSeqSup* value. The third and fourth lines respectively indicate the average sequential support and the average sequential confidence of the sequential rules found for the lowest *minSeqSup* value.

We found that the good performance of CMDeo for BMS-Webview1 and Toxin-Snake can be largely attributed to the fact that rule expansions done by CMDeo resulted in a larger proportion of valid rules than for Kosarak-1 and Kosarak-2 (line 2 of Table 2). For example, up to 43.56 % of rule expansions for Toxin-Snake resulted in valid rules, but less than three percent for Kosarak-1.

For CMRules, it was globally observed that the execution time grows proportionally to the number of association rules. Another observation is that the time for converting a sequence database in a transaction database (Step 1 of the CMRules algorithm) remained negligible during all experiments. CMRules' worst performance was for Toxin-Snake. CMRules did not perform well on Toxin-Snake because each item appears in almost every sequence and generally many times in the same sequence. Because of this and because of the relative positions of items in sequences, the percentage of association rules that are sequential rules was very low for this datasets. For example, for *minSeqSup* = 0.93, 204,986 association rules were found, and only 4,626 of those were valid sequential rules (about 2 % of all association rules). This means that CMRules had to check many sequential rules that are not sequential rules. This is very time-consuming because sup(X■Y) has to be computed for each of them. In the case of Toxin-Snake, calculating sup(X■Y) is particularly costly because the average sequential support of each sequential rule is high (e.g. the average sequential support is 95% for *minSeqSup* = 0.93). Therefore, for each association rule, CMRules had to check almost all sequences to calculate sup(X■Y).

In conclusion, we have noticed that the principal weakness of CMRules is its dependency on the number of association rules, while the main weakness of CMDeo is that for some datasets, rule expansions generate a large amount of invalid candidate rules. Also, it was shown that CMRules can benefit more from a high *minSeqConf* than CMDeo, and that both algorithms exhibited a linear scalability in terms of execution time with respect to the number of sequences |S|. The conclusion that we draw from these experiments is that both algorithms are more appropriate for certain datasets depending on their characteristics.

4.6 Choosing between CMRules and CMDeo

Because both algorithms perform better in some situations, an important question arises: "How to choose between the two algorithms?". To choose between the two algorithms several aspects should be considered.

The first aspect is how difficult it is to implement the algorithms. For implementation, one may prefer CMRules because it is an extension of existing association rule mining algorithms. Therefore, if one possesses an implementation of an association rule mining algorithm, it is probably easier to implement CMRules than implementing CMDeo from scratch. Also, if for a given application, a user wishes to discover both association rules and sequential rules, CMRules should be preferred because it can output both at the same time, whereas CMDeo only generates sequential rules.

The second aspect that has to be considered is the performance. If the performance is not an issue or a high *minSeqSup* threshold is used, then any algorithm could be used (both algorithms have similar performance under high *minSeqSup* thresholds). If performance is an issue, then the dataset's characteristics should be considered. We identified three main characteristics that influence the performance: (1) how dense a dataset is, (2) the percentage of association rules that are sequential rules and (3) the percentage of rule expansions that result in valid rules. Because the two last characteristics are hard to estimate without running the algorithms, we suggest to mainly consider if a dataset is sparse or dense for choosing between CMRules and CMDeo. To determine if a dataset is sparse or dense, one could look at the dataset's basic characteristics such as the number of sequences, number of items, average sequence length and the average number of items per sequence. For example, if a dataset contains many long sequences and the number of items is small, one may prefer CMDeo, while if the dataset contains many items and each one do not occur often in each sequence, one may prefer CMRules. The type of data contained in a dataset (e.g. customer data from a webstore) may also give indications about whether a dataset is dense or if it contains a large amount of association rules.

However, it would undoubtedly be preferable to have an automatic way of analyzing a dataset's characteristics to choose between CMDeo and CMRules. This is however a difficult research problem. In the literature, several researchers have worked on measures and formal methods for characterizing datasets (including for measuring their density) [25, 26, 27, 28], predicting the number of patterns that will be found by specific data mining algorithms [29] and understanding their distribution [23, 24]. However, these works mostly concerns transaction databases and the task of frequent itemset mining. To our knowledge, no such works have been done for sequence databases and sequential rule mining. Characterizing sequence databases and developing methods for predicting the number of rules found by different sequential rule mining algorithms is an open research problem.

Lastly, for applications where performance matters and where the selected algorithm will be run periodically on the same kind of data (for example, consider a system where the algorithm is applied everyday on customer data to generate a daily report), a reasonable approach is to run both algorithms once to choose the fastest. However, the disadvantage of this approach is that it requires implementing both algorithms.

5 Application of the Algorithm in an Intelligent Tutoring Agent

To illustrate the utility of CMRules, we briefly report its use in a real application (for more details, the reader is referred to [6]). The application is an intelligent tutoring agent named CTS [6], which assists learners during training sessions in virtual learning environments. CTS is integrated in CanadarmTutor [7] (cf. figure 12), a simulator-based tutoring system for learning to operate the Canadarm2 robotic arm, installed on the international space station. During training sessions, CTS interacts with the learner by offering hints, questions, encouragements and explanations.

CTS select interactions according to scenarios specified by domain experts into a structure called the "Behavior Network" [6]. To generate custom interactions adapted to each learner based on the predefined scenarios, CTS is equipped with different types of learning mechanisms (e.g., episodic and causal). These mechanisms allow CTS to adapt its behavior to each learner. We refer the reader to articles describing CTS [6, 30] for full details about its architecture and its learning mechanisms.

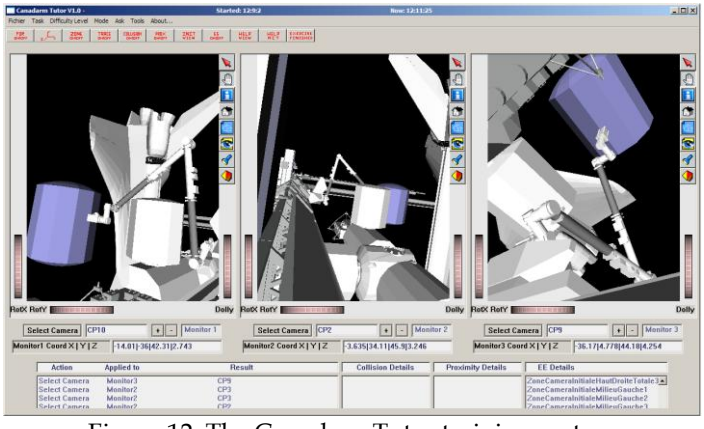

Figure 12. The CanadarmTutor training system

We have integrated the CMRules algorithm in CTS to give it the capability of finding the cause of learners' mistakes (the ability of finding causes is referred as "causal learning" in the cognitive science literature [31, 32, 33]). To do so, we have modified CTS so that it records each of its executions (interactions with learners during a training session) as a sequence in a sequence database. Sequences are ordered by time and each sequence's itemset contains a single item representing an event. An event represents the perception of a specific situation in the virtual world (e.g., a collision between Canadarm2 and the space station) or an interaction with the learner. Sequences usually contains from 20 to 40 itemsets depending on the length of interactions with learners.

To discover rules from these sequences, we have modified CTS so that it applies CMRules after each execution to discover sequential rules common to all the executions. In our implementation, we set *minSeqConf* =0.3 and *minSeqSup* =0.3 to find rules. We choose low support thresholds so that more rules are found. Example of rules that have been found are "the learner makes the robotic arm pass too close to the space station ⇒ the learner move the wrong joint, the learner is not aware of the distance" and "the learner is inactive ⇒ the learner lacks motivation".

Once rules have been extracted, CTS uses them in its next executions to guess the cause(s) of the learner's mistakes and the reason(s) of the learner behavior in general. To do this, we have integrated in CTS a mechanism that works as follows. To find the cause(s) of learner's mistakes, CTS compares the sequence of events from its current execution with the sequential rules found in its previous executions. CTS notes each rule that matches with the current execution. A rule is said to match if all items from its left part appears in the current execution and at least one item from its right part correspond to cause(s) that CTS want to guess in the current situation. For example, if a learner moves Canadarm2

too close to the space station (which is a dangerous situation, because of collision risks), CTS must determine the cause(s) of this situation to take an action. In this example, several causes are possible. One of them is that the learner did not select the correct joint of Canadarm2. This could be represented by the following rule "the learner makes Canadarm2 pass too close to the space station ⇒ the learner selected the wrong joint ".

Another possible cause is that the learner did not adjust a camera correctly, thus resulting in a poor vision, which could be represented by the following rule "the learner makes Canadarm2 pass too close to the space station ⇒ the learner did not adjust a camera correctly". If several such rules matches with the current execution, CTS has to choose one. In our implementation, we have programmed CTS to choose the rule with the highest *strength*. We define the *strength* of a rule X⇒Y as *Strength*(X⇒Y) = *seqConf*(X⇒Y) × *seqSup*(X⇒Y). We chose this measure as it seems to be a good way of taking into account both the sequential confidence and the sequential support (note that other criteria could be used). Then, according to the cause(s) of the mistake indicated by the selected rule, CTS will intervene to help learners to prevent future mistakes. For example, if the selected rule indicates that the cause is that the learner selected the wrong joint, CTS will invite the learner to justify why s/he chose that joint. If the learner answers with a proper justification, then CTS will consider that this is not the cause and select the next rule having the most strength. The aforementioned process continue until the cause(s) are found. If the cause(s) are successfully identified, CTS will record them in its current execution. In this way, CTS can learn to predict the causes of learners' mistakes and the reasons of their behavior in particular situations. Note that when no rule matches with the current execution, CTS makes random choices to guess the causes.

5.1 Evaluation

To evaluate the performance of CMRules in this application and assess how the new learning mechanism influences the performance of learners, we did an empirical evaluation. We asked eight users to test CTS with the new learning mechanism (version A) and CTS without the new learning mechanism (version B). Learners were invited to manipulate Canadarm2 for approximately 1 hour, using both versions A and B of the system. The first four students (Group 1) used version A, and then version B. The second four learners (Group 2), first used version B and then version A. During the experiments with version B, CTS recorded 363 sequences with an average length of 23 itemsets. CMRules algorithm was executed after each execution starting from the fourth execution. On average, sequential rules represented 31.05 % of all association rules, and the average number of sequential rules was 27. The execution time of the algorithm was short (less than 80 ms). After the experiment, we also applied CMDeo on the same sequence database to compare its performance with CMRules. CMDeo was about 10 ms slower and the memory usage was similar (about 20 megabytes). The reason why CMRules performed slightly better is that although sequences are long, the sequence database is not dense, and the percentage of association rules that are sequential rules is high. Totally, more than 2,400 rules were found from interactions with learners. To verify the quality of the rules found by CTS, we asked a domain expert to evaluate them. Given that checking all rules one by one would be tedious, the expert examined 150 rules from the 2,400+ recorded rules. Overall, the expert confirmed the correctness of about 85 % of the rules.

To evaluate how the new version of CTS influences the performance of learners using CanadarmTutor, we measured four performance criteria during the empirical evaluation for Group 1 and Group 2. Those are (1) the percentage of questions answered correctly (2) the average time to complete an exercise (3) the average number of collision risks provoked by learners, and (4) the average number of protocol violations. Overall, Group 1 performed better than Group 2 on all four criteria. For criterion 1, the percentage for Group 1 was 30 %, whereas it was 50% for Group 2. For criterion 2, the average time to complete an exercise was 1.4 and 2.2 minutes for Group 1 and Group 2, respectively. For criterion 3, the average number of collision risks was 3 and 5 for Group 1 and Group 2, respectively. For criterion 4, the protocol violation counts were 10 and 18, respectively. Although the number of learners who participated in this experiment is not very large, from these results, we can see that the version of CTS integrating CMRules improved their performance.

Furthermore, we analyzed the correctness of the CTS hints and messages to the learners during tasks. To determine if an intervention was correct, we asked learners to rate each CTS interventions as being appropriate or inappropriate. Because learners could incorrectly rate the tutor's interventions, we also observed the training sessions and verified the ratings given by each learner, subsequently. Results show that the average number of appropriate interventions was about 83 % using version 1 and 58 % using version 2. This is a notable improvement in CTS performance.

We also assessed the user satisfaction by performing a 10 minute post-experiment interview with each user. We asked each participant to tell us which version of CTS they preferred, to explain why, and to tell us what should be improved. Users unanimously preferred the new version. Some comments given by users were: 1) the tutoring agent "exhibited a more intelligent and natural behavior"; 2) the "interactions were more varied"; 3) the "tutoring agent seems more flexible"; 4) "in general, it gives a more appropriate feed-back". There were also several comments on how CTS could be improved. In particular, many users expressed the need to give CTS a larger knowledge base for generating dialogues. We plan to address this issue in future work.

5.2 Why using CMRules or CMDeo instead of another sequential rule mining algorithm?

An important question is: "Why using CMRules or CMDeo in CTS instead of another sequential rule mining algorithm?". First, as mentioned in the introduction, most sequential rule mining algorithms discover rules in a single sequence. In CTS, we aimed at discovering rules common to several executions. For this reason, algorithms that mine rules from a single sequence (e.g. [5, 8, 11]) and algorithms that do not discover rules common to several sequences (e.g. [4, 9]) could not be applied.

To our knowledge, the only algorithms from the literature that meets the requirement of mining sequential rules common to several sequences are the algorithms of Lo et al. [14], Pitman and Zanker [36] and Zhao et al. [34, 35]. They discover rules of the form X⇒Y, where X and Y are sequential patterns (ordered lists of itemsets). However, as we have explained in the introduction, this form of rules is too restrictive because if the items in X or in Y are not ordered in the same way in a sequence, it will be counted as an occurrence of a distinct rule. This can cause several problem. First, some similar rules may have important differences in how they are ranked according to interestingness measures (e.g. confidence and support. Second, some rules may not be discovered because they are considered uninteresting unless they would be merged with other rules. Third, because the rules are very specific, they are less likely to match with the sequences of actions in CTS, and therefore to be used for prediction.

In the case of CTS, the order of elements in X or Y is not considered important. We only wish to know what is the cause(s) of events, without considering the relative ordering of the cause(s) and of the events. Those are the reasons why we have not applied algorithms of Lo et al., Pitman and Zanker or Zhao et al. in CTS. We instead decided to define a more general form of sequential rules where items in the left part and in the right part of each rule are unordered.

6 Conclusion

This article presented CMRules, a correct and complete algorithm for mining sequential rules common to several sequences in sequence databases. The algorithm is designed to mine a less restrictive form of rule than the one discovered by the algorithms of Lo et al. [14], Pitman and Zanker [36] and Zhao et al. [34, 35]. CMRules is based on association rule mining because it was observed that for some datasets, the number of association rules that are sequential rules is high (this was confirmed in the experiments). Because CMRules is based on association rule mining, it is very efficient for discovering both association rules and sequential rules at the same time.

CMRules can be extended in several ways. First, it is possible to give additional capabilities to CMRules by implementing it based on an association rule mining algorithm having these capabilities. For instance, if one implements CMRules by using an incremental association rule mining algorithm such as the one of Cheung et al. [3], the result would be an incremental algorithm for mining sequential rules. A second example is to use a parallel algorithm for mining association rules such as the one of Agrawal & Shafer [15] to obtain a parallel algorithm for mining sequential rules.

Besides this type of extensions, it is also possible to extend CMRules by modifying the code for checking if a rule is included in a sequence. For example, one could easily modify the definition of sup(X■Y) so that the antecedent and consequent of a rule have to occur within a given time frame, as it is proposed in the works of Mannila et al. [13], Das et al. [4] and Harms et als. [9]. Another simple extension of CMRules would be to mine rules respecting constraints on the minimal and maximal size of antecedents and consequents (*minAntecedentSize, maxAntecedentSize, minConsequentSize, maxConsequentSize*). To do this extension, two simple changes needs to be done. The first change is to mine only itemsets of size *maxAntecedentSize* + *maxConsequentSize* during Step 2 of the algorithm. Mining only these itemsets is sufficient for finding all sequential rules because association rules are generated by combining pair of frequent itemsets $X \subseteq Y$ to get rule of the form X→Y-X [1]. The second change that needs to be done is to how the rules are generated from frequent itemsets that are discovered. In the Agrawal et al.algorithm [1], rules are constructed recursively, each recursion growing or shrinking the left or right part of a rule, one item at a time. It is easy to incorporate a check in this process to stop growing/shrinking rules if it will not respect size constraints. For CMDeo, similar extensions can also be done. For example, it is easy to add rule size constraints, because rules are also expanded one item at a time. However, only CMRules can benefit from extensions by the reuse of existing association rule mining algorithms with particular capabilities.

In this article, we evaluated the performance of CMRules in three different ways. We first analyzed its time complexity. Second, we compared its performance on real datasets with an adaptation of the Deogun et al.'s algorithm [5] that we named CMDeo. For this comparison, we chose datasets that are commonly used in the data mining literature and have different properties. These experiments have shown that both CMRules and CMDeo perform better depending on each dataset's characteristics. In particular, CMRules was shown to be faster for datasets where the percentage of association rules that are sequential rules is high, whereas CMDeo is faster for dense datasets where rule expansions generate a large proportion of valid rules.

Thirdly, we have described how the algorithm is used in an intelligent tutoring agent that assist learners during

training sessions with the Canadarm2 robotic arm installed on the International Space Station [6]. Thanks to the algorithm, the agent can now learn sequential relationships between events common to several of its executions. The agent exploits this information, to guess the cause(s) of the learner's mistakes. Note that we have also applied the CMRules algorithm in another tutoring agent for procedural knowledge learning [16].

ACKNOWLEDGEMENTS

The authors thank the Canadian Space Agency, the Fonds Québécois de la Recherche sur la Nature et les Technologies and the Natural Sciences and Engineering Research Council for their logistic and financial support. The authors also thank members of GDAC/PLANIART teams who have participated in the development of CanadarmTutor.

REFERENCES

[1] R. Agrawal, T. Imielminski and A. Swami, "Mining Association Rules Between Sets of Items in Large Databases", *Proc. 1993 ACM SIGMOD International Conference on Management of Data*, pp. 207-216, ACM Press, 1993.

[2] R. Agrawal and R. Srikant, "Mining Sequential Patterns", *Proc. Eleventh International Conference on Data Engineering*, pp. 3-14, IEEE Computer Society, 1995.

[3] D.W. Cheung, J. Han, V. Ng and Y. Wong, "Maintenance of discovered association rules in large databases: An incremental updating technique", *Proc. Twelfth International Conference on Data Engineering*, pp. 106-114, IEEE Computer Society, 1996.

[4] G. Das., K.-L. Lin, H. Mannila, G. Renganathan and P. Smyth, "Rule Discovery from Time Series". *Proc. Fourth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 16-22, ACM Press, 1998.

[5] J.S. Deogun and L. Jiang, "Prediction Mining – An Approach to Mining Association Rules for Prediction". *Proc. Tenth International Conference on Rough Sets, Fuzzy Sets, Data Mining, and Granular Computing*, pp. 98-108, Springer, 2005.

[6] U. Faghihi, P. Fournier-Viger, R. Nkambou and P. Poirier, "The Combination of a Causal Learning and an Emotional Learning Mechanism for an Improved Cognitive Tutoring Agent", *Proc. Twenty third International Conference on Industrial, Engineering and Other Applications of Applied Intelligent Systems*, pp. 438-449, Springer, 2010.

[7] F. Kabanza, R. Nkambou and K. Belghith, "Path-planning for Autonomous Training on Robot Manipulators in Space", *Proc. Nineteenth International Joint Conference on Artificial Intelligence*, pp. 35-38, Professional Book Center, 2005.

[8] H. J. Hamilton and K. Karimi, "The TIMERS II Algorithm for the Discovery of Causality". *Proc. Ninth Pacific-Asia Conference on Knowledge Discovery and Data Mining*, pp. 744-750, Springer, 2005.

[9] S. K. Harms, J. Deogun and T. Tadesse, "Discovering Sequential Association Rules with Constraints and Time Lags in Multiple Sequences", *Proc. ACM SIGART Thirteenth International Symposium on Methodologies for Intelligent Systems*, pp. .373-376, ACM Press, 2003.

[10] M. Hegland, "The Apriori Algorithm – A Tutorial", *Mathematics and Computation in Imaging Science and Information Processing*, vol. 11, pp. 209-262, World Scientific Publishing Co., 2007.

[11] Y. L. Hsieh, D.-L. Yang and J. Wu, "Using Data Mining to Study Upstream and Downstream Causal Relationship in Stock Market". *Proc. Ninth Joint Conference on Information Sciences*, Atlantis Press, 2006.

[12] S. Laxman and P. Sastry, "A survey of temporal data mining", *Sadhana*, vol. 3, pp. 173-198, Springer, 2006.

[13] H. Mannila, H. Toivonen and A.I. Verkano, "Discovery of frequent episodes in event sequences", *Data Mining and Knowledge Discovery*, vol. 1, no. 1, pp. 259-289, Springer, 1997.

[14] D. Lo, S.-C. Khoo and L. Wong, "Non-redundant sequential rules - Theory and algorithm", *Information Systems*, vol. 34, no. 4-5, pp. 438-453, Wiley-Blackwell, 2009.

[15] R. Agrawal and J.C. Shafer, "Parallel Mining of Association Rules", *IEEE Transactions on Knowledge and Data Engineering*, vol. 8, no. 6, pp. 962-969, IEEE Computer Society, 1996.

[16] P. Fournier-Viger, "Knowledge Discovery in Problem-Solving Learning Activities", Ph.D. thesis, University of Quebec at Montreal, 2010.

[17] P. Fournier-Viger, U. Faghihi, R. Nkambou and E. Mephu Nguifo, "CMRules: An Efficient Algorithm for Mining Sequential Rules Common to Several Sequences", *Proc. Twenty-Third International Florida Artificial Intelligence Research Society Conference*, pp. 410-415, AAAI Press, 2010.

[18] R. Nkambou, P. Fournier-Viger and E. Mephu Nguifo, "Learning Task Models in Ill-defined Domain Using an Hybrid Knowledge Discovery Framework", *Knowledge-Based Systems*, Elsevier Science, vol. 24, no. 1, pp. 176-185, Elsevier, 2011.

[19] I. Jonassen, J.F. Collins and D. G. Higgins, "Finding flexible patterns in unaligned protein sequences", *Protein Science*, vol. 4, no. 8, pp. 1587–1595, Wiley-Blackwell, 1995.

[20] J. Pei, J. Han, B. Mortazavi-Asl, J. Wang, H. Pinto, Q. Chen, U. Dayal and M.-C. H, "Mining Sequential Patterns by Pattern-Growth: The PrefixSpan Approach", *IEEE Transactions on Knowledge and Data Engineering*, vol. 16, no. 10, pp.1-17, IEEE Computer Society, 2004.

[21] M. J. Zaki, "SPADE: An Efficient Algorithm for Mining Frequent Sequences", *Machine Learning*, vol. 42, no. 1-2, pp. 32-60, Springer, 2001.

[22] J. Ayres, J. Flannick, J. Gehrke and T. Yiu, "SPAM: Sequential Pattern mining using a bitmap representation", *Proc. Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 429-435, ACM Press, 2002.

[23] L. Lhote, F. Rioult and A. Soulet, "Average of frequent and closed patterns in Random Databases", *Proc. Conférence francophone sur l'apprentissage automatique* 2005, pp. 345-360, Presses Universitaires de Grenoble, 2005.

[24] R. Emilion and G. Lévy, "Size of random Galois lattices", *Discrete Applied Mathematics*, no. 157, pp. 2945-2057, Elsevier, 2009.

[25] A. Salleb-Aouissi and C. Vrain, "A Contribution to the Use of Decision Diagrams for Loading and Mining Transaction Databases", *Fundamentae Informaticae*, vol. 78, no. 4, pp. 575-594, IOS Press, 2007.

[26] A. Salleb and C. Vrain, "Estimation of the Datasets Density Relying on Decision Diagrams", *Proc. 15th International Symposium on Methodologies for Intelligent Systems*, pp. 688-697, Springer, 2005.

[27] F. Flouvat, F. De Marchi and J.-M. Petit, "A New Classification of Datasets for Frequent Itemsets", *Journal of Intelligent Information Systems*, vol. 34, p. 1-19, Springer, 2010.

[28] T. Hamrouni, S. Ben Yahia and E. Mephu Nguifo, "Towards a Finer Assessment of Extraction Contexts Sparseness", *Proc. 8th International Workshop on Database and Expert Systems Applications*, pp. 504-508, IEEE Computer Society, 2007.

[29] F. Geerts, B. Goethals and J.V. den Bussche, "A Tight Upper Bound on the Number of Candidate Patterns", *Proc. 2001 IEEE International Conference on Data Mining*, pp. 155-162, IEEE Computer Society, 2001.
[30] U. Faghihi, P. Poirier, P. Fournier-Viger and R. Nkambou, "Human-Like Learning in a Cognitive Agent", *Journal of Experimental and Theoritical Artificial Intelligence*, Taylor and Francis, in press..
[31] A. Gopnik, C. Glymour, D. M. Sobel, L. E. Schulz, T. Kushnir and D. Danks, "A Theory of Causal Learning in Children: Causal Maps and Bayes Nets", *Psychological Review*, vol. 111, no. 1., p. 3-32, American Psychological Association, 2004.
[32] A. Gopnik and L. Shulz, *Causal Learning: Psychology, Philosophy and Computation*, Oxford University Press, 2007.
[33] A. Maldonado, A. Catena, J. C. Perales and A. Candido, "Cognitives Biases in Human Causal Learning", *Spanish Journal of Psychology*, vol. 10, no. 2, p. 242-250, U. Complutense de Madrid, 2007.
[34] Y. Zhao, H. Zhang, L. Cao, C. Zhang and H. Bohlscheid, "Mining Both Positive and Negative Impact-Oriented Sequential Rules From Transactional Data", *Proc. of the 13th Pacific-Asia Conference on Knowledge Discovery and Data Mining*, pp.656-663, Springer, 2009
[35] Y. Zhao, H. Zhang, L. Cao, C. Zhang and H. Bohlscheid, "Efficient Mining of Event-Oriented Negative Sequential Rules", *Proc. of the 2008 IEEE/WIC/ACM International Conference on Web Intelligence*, pp. 336-342, IEEE Computer Society, 2008.
[36] A. Pitman and M. Zanker, "An Empirical Study of Extracting Multidimensional Sequential Rules for Personalization and Recommendation in Online Commerce". *Proc. Wirtschaftinformatik 2011*, pp.180-189, AIS Electronic Library, 2011.

Philippe Fournier-Viger (Ph.D.) is a post-doctoral researcher in Data Mining at National Cheng Kung University, in Taiwan. He received a Ph.D. in Cognitive Computer Science at the University of Quebec in Montreal (2010). He is a member of the IEEE, AACE and AIED societies. His work is funded by a FQRNT Canadian Graduate Scholarship grant. His research interests include data mining, e-learning, intelligent tutoring systems, knowledge representation and cognitive modeling. He is currently working on several data mining projects.

Usef Faghihi received his Ph.D. (2010) in Cognitive Computer Science at the University of Quebec in Montreal. He is a post-doctoral researcher at the Cognitive Computing Research Group (CCRG) laboratory at University of Memphis. He is also a member of the GDAC research team and the IEEE, AACE and JAMAS societies. His work is funded by a FQRNT Canadian Graduate Scholarship grant. His research interests include cognitive modeling, learning in cognitive agents, e-learning, intelligent tutoring systems, user modeling and data mining.

Roger Nkambou (Ph.D) is currently a Professor of Computer Science at the University of Quebec at Montreal, and Director of the GDAC (Knowledge Management Research) Laboratory (http://gdac.dinfo.uqam.ca). He received his Ph.D. (1996) in Computer Science from the University of Montreal. His research interests include knowledge representation, intelligent tutoring systems, intelligent software agents, ontology engineering, student modeling and affective computing. He also serves as member of the program committee of the most important international conferences in Artificial Intelligence and Education.

Engelbert Mephu Nguifo (Ph.D) is currently a Professor in Computer Science at University of Artois (Institute of Technology of Lens), France. He received his Ph.D (1993) in Computer Science from the University of Montpellier. He is a member of the Computer Science research center of Lens (CRIL-CNRS). His research interests are data mining, machine learning, bioinformatics, human-computer interaction, information visualization, knowledge representation and formal concept analysis. He is also member of the ACM society and ACM-SIGKDD group, of Canadian AI society, and French AI, bioinformatics, data mining, and classification associations (AFIA, SFBI, EGC and SFC).