

CMRULES: An Efficient Algorithm for Mining Sequential Rules Common to Several Sequences

Philippe Fournier-Viger¹, Usef Faghihi¹, Roger Nkambou¹, Engelbert Mephu Nguifo²

¹Department of Computer Sciences, University of Quebec in Montreal, 201, avenue du Président-Kennedy, Montréal, Canada

²Department of Mathematics and Computer Sciences, Université Blaise-Pascal Clermont 2, BP 125, 63173 Aubière, cedex, France
{fournier_viger.philippe, faghihi.usef}@courrier.uqam.ca, nkambou.roger@uqam.ca, mephu@isima.fr

Abstract

We propose CMRULES, an algorithm for mining sequential rules common to many sequences in sequence databases – not for mining rules appearing frequently in sequences. For this reason, the algorithm does not use a sliding-window approach. Instead, it first finds association rules to prune the search space for items that occur jointly in many sequences. Then it eliminates association rules that do not meet minimum confidence and support thresholds according to the time ordering. We evaluated the performance of CMRULES in three different ways. First, we provide an analysis of its time complexity. Second, we compared its performance on a public dataset with a variation of an algorithm from the literature. Results show that CMRULES is more efficient for low support thresholds, and has a better scalability. Lastly, we report a real application of the algorithm in a complex system.

Introduction

Discovering temporal relationships between events stored in large databases is important in many domains, as it provides a better understanding of the relations between events, and sets a basis for the prediction of events. For example, in international trade, one could be interested in discovering relations between the appreciation of currencies to make trade decisions. Various methods have been proposed for mining temporal relations between events in databases (see for example Laxman & Sastry, 2006, for a survey).

In the field of data mining, one of the most popular set of techniques for discovering temporal relations between events in discrete time series is sequential pattern mining (Agrawal & Srikant, 1995), which consists of finding sequences of events that appear frequently in a sequence database. However, knowing that a sequence of events appear frequently in a database is not sufficient for the prediction of events. For example, it is possible that some event y appears frequently after an event x but that there are also many cases where x is not followed by y . In this case, predicting that y will occur if x occurs on the basis of

a sequential pattern x,y could be a huge mistake. Thus, for prediction, it is desirable to have patterns that indicate how many times x appeared before y and how many times x appeared and y did 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 two, as in the previous example– and current algorithms have just not been designed for this.

The alternative to sequential pattern mining that address the problem of prediction is *sequential rule mining* (Mannila et al., 1997; Das et al., 1998; Harms et al., 2002; Hamilton et al., 2005; Hsieh et al., 2006; Deogun & Jiang, 2005). 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 (Das et al., 1998; Hsieh et al., 2006), weather observation (Hamilton & Karimi, 2005) and drought management (Harms et al, 2002; Deogun & Jiang, 2005).

The most famous approach for sequential rule mining is that of Mannila et al. (1997) and other researchers afterward that aim at discovering 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 sequential rules respecting a minimal confidence and support (Mannila et al., 1997). These rules are of the form $X \Rightarrow Y$, 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, their work 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 (2005), Hsieh et al. (2006) and Deogun & Jiang (2005), 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 works have been designed for mining sequential rules in several sequences (Das et al., 1998; Harms et al., 2002). For example, Das et al. (1998) discovers rules where the left part of a rule can have multiple events, yet the right part still has to contain a single event. This is a serious limitation, as in real-life

applications, sequential relationships can involve several events. Moreover, the algorithm of Das et al. (1998) is highly inefficient as it tests all possible rules, without any strategy for pruning the search space. To our knowledge, only the algorithm of Harms et al. (2002) 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 user-specified thresholds. The support of a rule is here defined as the number of times that the right part occurs after the left part within user-defined time windows.

However, one important limitation of the algorithms of Das et al., (1998) and Harms et al. (2002) comes from the fact that they are designed for mining rules occurring frequently in sequences. As a consequence, 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 the items 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. (1998) and Harms et al. (2002) are inappropriate 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. A second example is the application domain of this paper. We have built an intelligent tutoring agent that records a sequence of events for each of its executions. We wish that the tutoring agent discovers sequential rules between events, common to several of its executions, so that the agent can thereafter use the rules for prediction during its following execution.

In this paper, we address this problem by proposing an algorithm specifically designed for mining sequential rules common to many sequences, and that does not restrict the number of events in each rule. Because it has a different purpose, the algorithm does not rely on a sliding-window approach. Instead it finds associations rules between items to prune the search space to items that occur jointly in many sequences. Then it eliminates association rules that do not meet minimum confidence and support thresholds according to the time ordering. In this paper, we prove that this correctly discovers all sequential rules if parameters for mining association rules are chosen correctly.

The paper is organized as follows. The next section gives a brief overview of the problem of association rule mining, defines the problem of mining sequential rules common to many sequences, and analyzes the relation between sequential rules and association rules. Next the third section presents our algorithm based on association rule mining, a proof that if the confidence and support are chosen correctly the set of all sequential rules is found, and an analysis of the algorithm's time complexity. The fourth section presents an evaluation of the algorithm with a real dataset and an application in a complex system for providing help to students during learning activities. Finally, the last section draws a conclusion.

Background and Problem Definition

Association rule mining (Agrawal et al., 1993) 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 $T = \{t_1, t_2, \dots, t_n\}$ and a set of items $I = \{i_1, i_2, \dots, i_n\}$, where $t_1, t_2, \dots, t_n \subseteq I$. The support of an itemset $X \subseteq I$ for a database is denoted as $\text{sup}(X)$ 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 (Agrawal et al. 1993) 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 $\text{sup}(X \cup Y) / |T|$. The confidence of a rule is defined as $\text{conf}(X \rightarrow Y) = \text{sup}(X \cup Y) / \text{sup}(X)$. Since $|T| \geq \text{sup}(X)$ for any $X \subseteq I$, the relation $\text{conf}(r) \geq \text{sup}(r)$ hold for any association rule r .

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 (Agrawal & Srikant, 1995). A sequence database SD is defined as a set of sequences $S = \{s_1, s_2, \dots, s_n\}$ and a set of items $I = \{i_1, i_2, \dots, i_n\}$, where each sequence s_x is an ordered list of transactions $s_x = \{X_1, X_2, \dots, X_n\}$ such that $X_1, X_2, \dots, X_n \subseteq I$.

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 . We define two interestingness measures for such a rule, which are an adaptation for multiple sequences of the measures used for other sequential rule mining algorithms (Mannila et al., 1997; Das et al., 1998; Harms et al., 2002). The first measure is the rule's sequential support and is defined as: $\text{seqSup}(X \Rightarrow Y) = \text{sup}(X \blacksquare Y) / |S|$. The second measure is the rule's sequential confidence and is defined as: $\text{seqConf}(X \Rightarrow Y) = \text{sup}(X \blacksquare Y) / \text{sup}(X)$. Here, the notation $\text{sup}(X \blacksquare Y)$ 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 $\text{sup}(X)$ represents the number of sequences that contains X . Since $|S| \geq \text{sup}(X)$ for any $X \subseteq I$, the relation $\text{seqConf}(r) \geq \text{seqSup}(r)$ holds for any sequential rule r . We define the problem of mining sequential rules common to several sequences as the problem of finding all sequential rules from a sequence database such that their sequential support and sequential confidence are respectively higher or equal to some user-defined threshold *minSeqSup* and *minSeqConf*. As an example, figure 1 shows a sequence

database containing four transactions and some sequential rules found with $minSeqSup = 0.5$ and $minSeqConf = 0.5$.

ID	sequences	ID	rule	S-Sup.	S-Conf.
1	(a b), (c), (f), (g), (e)	1	abc ⇒ e	0.5	1.0
2	(a d), (c), (b), (e f)	2	a ⇒ cef	0.5	0.66
3	(a), (b), (f) (e)	3	ab ⇒ ef	0.5	1.0
4	(b), (f g)	4	b ⇒ ef	0.75	0.75
		5	a ⇒ ef	0.75	1.0
		6	c ⇒ f	0.5	1.0
		7	a ⇒ b	0.5	0.66
	

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

Note that the problem of mining sequential rules is largely different from the one of mining sequential patterns, since there is no time ordering inside itemsets of sequential rules. Therefore it is not possible to simply adapt a sequential pattern mining algorithm for generating these sequential rules.

Instead, the algorithm that we proposed 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 $r: X \Rightarrow Y$ of S has a corresponding association rule $r': X \rightarrow Y$ in S'. Since $\text{sup}(X \sqcap Y)$ is always lower or equal to $\text{sup}(X \cup Y)$ the relationships (1) $\text{sup}(r') \geq \text{seqSup}(r)$ and $\text{conf}(r') \geq \text{seqConf}(r)$, hold for any sequential rule r and its corresponding association rule r'.

An Algorithm for Mining Sequential Rules

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

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 (Agrawal et al., 1993). 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. $\text{seqSup}(r) < minSeqSup$
b. $\text{seqConf}(r) < minSeqConf$
4. Return the set of rules

Figure 2: The CMRULES algorithm

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). Then, CMRULES calculates the sequential support and sequential confidence of each association rule by scanning the sequence database, and then eliminate 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.

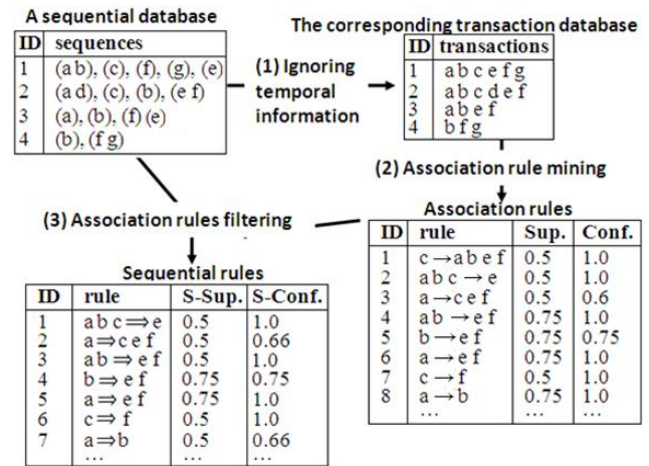


Figure 3: A sample execution of the CMRULES algorithm

Proof of Completeness

We now show that the CMRULES can find all sequential rules. To prove this, we have to demonstrate that the set of association rules contains the set of all sequential rules. This is proven next.

Theorem 1. The algorithm discovers all sequential rules for a given $minSeqConf$ and $minSeqSup$ if we choose $minconf \leq minSeqConf$ and $minsup \leq minSeqSup$.

Proof. To prove this, we can consider the sequential support and sequential confidence interestingness measures separately, without loss of generality.

Let's first consider the sequential support. Suppose there is a sequential rule r such that $\text{seqSup}(r) \geq minSeqSup$ and $minsup > \text{sup}(r')$ (we suppose that the corresponding association rule r' is not frequent). We know that $\text{sup}(r') \geq \text{seqSup}(r)$. Therefore, it follows that $minsup > \text{sup}(r') \geq \text{seqSup}(r) \geq minSeqSup$. Thus, $minsup \geq minSeqSup$. Therefore, if we choose $minSeqSup \geq minsup$ there will be no such rule r .

Now consider the sequential confidence. Suppose there is a sequential rule r such that $\text{seqConf}(r) \geq minSeqConf$ and that $minconf > \text{conf}(r')$. We know that $\text{conf}(r') \geq \text{seqConf}(r)$. Therefore $minconf > \text{conf}(r') > \text{seqConf}(r) \geq minSeqConf$. Therefore, if we choose $minSeqConf \geq minconf$ 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*.

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 $X \rightarrow Y$ is contained in it to calculate the $\text{sup}(X \blacksquare Y)$ value that is necessary for calculating the sequential confidence and sequential support. Checking if a rule is contained 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, checking if each rule is contained in each sequence is inefficient. 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 generate association rules, one needs to apply an algorithm such as Apriori (Agrawal et al., 1993). Algorithms for association rule mining proceed in two steps (Agrawal et al., 1993). 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 $X \subseteq Y$, to generate rules of the form $X \rightarrow Y - X$ (see Agrawal et al., 1993 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 (Agrawal et al., 1993) (in the case of Apriori, this variation is known as Apriori-TID; cf. Agrawal et al., 1993). Having this extra information about frequent itemsets, Step 3 can be performed efficiently by checking each rule $X \rightarrow Y$ 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 $\text{sup}(X \blacksquare Y)$ can be calculated by checking only with the sequences containing X (by the definition of $\text{sup}(X \blacksquare Y)$, and the other terms for calculating the sequential support and confidence ($\text{sup}(X)$ and $|S|$) are known. In our tests, this simple optimization improved step 3's performance by up to 50%.

Reducing Memory Consumption

It is possible to merge Step 2 and Step 3 of CMRULES so that each association rule that is generated is immediately checked for its sequential support and confidence. This allows considering only one association rule at a time in central memory. Therefore, the set of all association rules does not need to be stored. 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 also be written to the disk at the same time. This implementation strategy greatly reduces the memory requirement of the algorithm.

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 of) is linear with respect to the number of sequences and their size.

The time complexity of Step 2 is more difficult to establish. It depends on the association rule mining algorithm that is used. As explained previously, mining association rule is done in two steps: mining frequent itemsets and generating rules. The first step is the most costly (Agrawal et al., 1993), so it is adequate to ignore the second step for estimating time complexity. If we use the Apriori algorithm for mining frequent itemsets, the time complexity is $O(d^2 n)$ where d is the number of different items and n is the number of transactions in the database (see Hegland, 2007 for a proof). In our implementation, we used Apriori-TID a variation that performs slightly better for some datasets (Agrawal et al., 1993) and has a similar time complexity (Hegland, 2007).

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 $|S| \times \text{minsup}$ sequences, and $|S|$ sequences to be checked for each rule. Checking if a rule is contained 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.

Evaluation of the Algorithm

We have implemented CMRULES in the Java programming language. We describe next its evaluation with a real dataset, and an application of the algorithm.

Evaluation of the Algorithm with the Kosarak Dataset

To evaluate the performance of the algorithm, we compared its performance with a second algorithm on a real dataset. The second algorithm is the one of Deogun et al. (2005). Although this latter was proposed for mining sequential rules in a single sequence, it is easy to adapt it

for the case of mining rules common to multiple sequences. We have adapted it, and implemented it in the Java programming language. Since space is limited, we will only briefly describe our adaptation.

The algorithm proceeds as follow. It first finds all rules where the left and right sides contain a single item. This is done by counting the support of each items of size one by scanning the sequence database one time. Then, for each pair of frequent items x, y appearing in at least $minSeqSup$ same sequences, the algorithm generate candidates rules $\{x\} \Rightarrow \{y\}$ and $\{y\} \Rightarrow \{x\}$. These rules are then evaluated by calculating their sequential support and confidence to see if they respect $minSeqSup$ and $minSeqConf$. Calculating the sequential support and confidence is done exactly as in Step 3 of our algorithm. Once, these rules are found, the algorithm then recursively find larger candidate rules by combining rules of smaller size in a level-wise manner (similar to Apriori). This is done by two separate processes. Left-side expansion is the process of taking two candidate rules $X \Rightarrow Y$ and $Z \Rightarrow Y$, where X and Z are itemsets of size n sharing $n-1$ items, to generate a new larger candidate rule $XUZ \Rightarrow Y$. Right-side expansion is the process of taking two candidate rules $Y \Rightarrow X$ and $Y \Rightarrow Z$, where X and Z are itemsets of size n sharing $n-1$ items, to generate a new larger candidate rule $Y \Rightarrow XUZ$. After candidate rules are generated by left/right side expansion their sequential support and confidence are calculated by scanning sequences. To prune the search space for candidate rules, it can be shown easily that expanding the left side of a rule not respecting $minSeqSup$ will not result in valid sequential rules, and that expanding the right side of a rule not respecting $minSeqSup$ or $minSeqConf$ will not generate valid sequential rules. Also, by using a lexicographic ordering of items in itemsets, it can be shown that no candidate rule will be generated twice. It is important to note that the algorithm of Deogun et al. is designed to find a subset of all the sequential rules (that is not a lossless representation of all sequential rules), and that we have adapted it to find all sequential rules.

To compare the performance of both algorithms, we have used Kosarak, a public dataset available from <http://fimi.cs.helsinki.fi/data/>. It contains 990 000 click-stream data from the logs of online news portal. In a first experiment, we selected the 70 000 first sequences of the dataset. These sequences contain an average of 7.97 items each, from a set of 21 144 different items. We applied the algorithms on this dataset with $minSeqConf = 0.3$ and with $minSeqSup = 0.02, 0.019, \dots, 0.003$.

Figure 4.a shows the time required for running CMRULES and the time for executing the adaptation of the Deogun et al. algorithm (Algo 2). From this figure we can see that for high support, both algorithms have similar performance. But for lower support thresholds CMRULES is much faster. For example, whereas for $minSeqSup = 0.02$ both algorithms provides similar performance, for $minSeqSup = 0.004$ our algorithm is more than five times faster. This shows that CMRULES has a better scalability.

During the experiment, we also measured the number of association rules and sequential rules discovered for each value of $minSeqSup$. Figure 4.b illustrates these results. The number of sequential rules ranged from 11 to 20 % of all the association rules. This shows that the percentage of sequential rules can be quite high in real datasets. For example, for a $minSeqSup$ of 0.02, there is 312 association rules and 64 sequential rules (20.51 %), and for a $minSeqSup$ of 0.003, there is 21 947 association rules and 2 512 sequential rules (11.44%). We also observed that execution time of our algorithm seems to grow proportionally to the number of association rules. Lastly, we observed that the time for converting the sequence database in a transaction database (step 1 of our algorithm) is negligible (around 300 ms). We have performed other experiments on Kosarak that have shown the same trends. Due to space limitation they are not described in this paper.

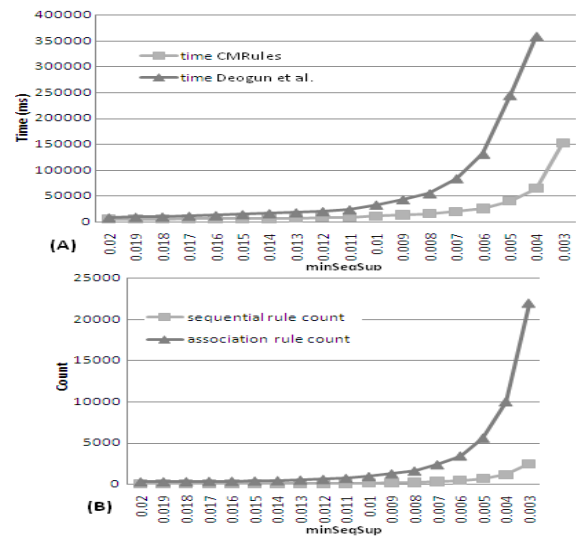


Figure 4: Result of experiment with the Kosarak dataset

Application of the Algorithm in an Intelligent Tutoring Agent

To further evaluate the algorithm and demonstrate its utility, we used it for a real application. We integrated the algorithm in an intelligent tutoring agent named CTS (Faghihi et al., 2010), designed for providing assistance to learners during learning activities in virtual environments. CTS is currently used for CanadarmTutor (Kabanza et al., 2010), a simulator-based tutoring system for learning to operate the Canadarm2 robotic arm, deployed on the international space station. During training sessions, CTS interacts with learners through a dialogue comprising hints, questions and explanations. CTS takes pedagogical decisions by relying on rules encoded in a structure named “the Behavior Network”, authored by humans (Faghihi et al., 2010).

We have integrated the algorithm in CTS to provide it with the capability of learning relations between events. To do so, we have modified CTS so that it records each of its

executions in a sequence database. Each sequence contains the information that CTS perceived and the decisions that it took during one execution. We have also modified CTS so that it applies the algorithm after each execution to discover sequential rules common to several executions. We conducted a short experiment in which we performed 100 executions of CTS. During these executions, we responded to the questions posed by CTS. This resulted in a sequence database of 100 sequences with an average length of 25 itemsets. The 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 50 ms). CTS found several relevant sequential rules such as “the learner move the wrong joint \Rightarrow the learner makes the robotic arm pass too close to the space station” and “the learner lacks motivation \Rightarrow the learner is inactive”. CTS then used the learned rules to adapt its behavior by making suggestions to learners such as to review the arm manipulation procedure or by generating encouragements. Overall, we observed that the behavior of CTS improved. However, we have not yet measured quantitatively how this new version of CTS influences the learning of students. We briefly describe here this experiment to illustrate an application of the algorithm. The reader can refer to a full paper describing this application for more details (Faghihi et al., 2010).

Conclusion

This paper presented CMRULES, an algorithm for mining sequential rules common to several sequences in sequence databases.

Because the algorithm is based on association rule mining it is very efficient if one wishes to discover both association rules and sequential rules in a database, as both are discovered at the same time. Moreover, it is possible to enhance the capability of CMRULES by choosing a particular association rule mining algorithm having this capability. For instance, if one implements CMRULES by using an incremental association rule mining algorithm such as the one of Cheung et al. (1996), the result would be an incremental algorithm for mining sequential rules. Besides this type of extensions, many others are possible by modifying the code for checking if a rule is included in a sequence. For example, one could easily modify the definition of $\text{sup}(X \blacksquare Y)$ 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. (1997), Das et al. (1998) and Harms et al. (2002).

We evaluated the performance of our algorithm in three different ways. We first analysed its time complexity. Second, we measured its performance with Kosarak, a large public dataset of click-streams data from the logs of an online news portal. With this experiment, we empirically observed that (1) our algorithm has a better scalability than the Deogun et al. algorithm adapted for the

same task and (2) that the time required for mining sequential rules seems to be proportional to the number of association rules. We can expect that our algorithm will perform better on datasets in which the percentage of association rules that are sequential rules is higher, as fewer rules will be rejected. On this point, it is encouraging that in our experiment this percentage was always between 10 and 20 %. Finally, we have presented an application of the algorithm in an intelligent tutoring agent. Thanks to the algorithm, the agent can now learn relationships between events common to several of its executions. This application illustrated one use of the algorithm. However, possible applications are numerous, as it can be applied to any sequence databases.

References

- Agrawal, R., Imielinski, T., & Swami, A. 1993. Mining Association Rules Between Sets of Items in Large Databases. *SIGMOD Conference*, 207-216
- Agrawal, R. & Srikant, R. 1995. Mining Sequential Patterns. *Proc. Int. Conf. on Data Engineering*, pp. 3-14.
- Cheung, D.W., Han, J., Ng, V. & Wong., Y. 1996. Maintenance of discovered association rules in large databases: An incremental updating technique. *Proc. ICDE 1996*, 106-114.
- Das., G., Lin, K.-I., Mannila, H., Renganathan, G., & Smyth, P. 1998. Rule Discovery from Time Series. *Proc. 4th Int. Conf. on Knowledge Discovery and Data Mining*.
- Deogun, J.S. & Jiang, L. 2005. Prediction Mining – An Approach to Mining Association Rules for Prediction. *Proc. of RSFDGrC 2005 Conference*, 98-108.
- Faghihi, U., Fournier-Viger, P., Nkambou, R. & Poirier, P., 2010. The Combination of a Causal Learning and an Emotional Learning Mechanism for Improved Cognitive Tutoring Agent. *Proceedings of IEA-AIE 2010* (in press).
- Kabanza, F., Nkambou, R. & Belghith, K. 2005. Path-planning for Autonomous Training on Robot Manipulators in Space. *Proc. 19th Intern. Joint Conf. on Artificial Intelligence*, 35-38.
- Hamilton, H. J. & Karimi, K. 2005. The TIMERS II Algorithm for the Discovery of Causality. *Proc. 9th Pacific-Asia Conference on Knowledge Discovery and Data Mining*, 744-750.
- Harms, S. K., Deogun, J. & Tadesse, T. 2002. Discovering Sequential Association Rules with Constraints and Time Lags in Multiple Sequences. *Proc. 13th Int. Symp. on Methodologies for Intelligent Systems*, pp. 373-376.
- Hegland, M. 2007. The Apriori Algorithm – A Tutorial. *Mathematics and Computation. Imaging Science and Information Processing*, 11:209-262.
- Hsieh, Y. L., Yang, D.-L. & Wu, J. 2006. Using Data Mining to Study Upstream and Downstream Causal Relationship in Stock Market. *Proc. 2006 Joint Conference on Information Sciences*.
- Laxman, S. & Sastry, P. 2006. A survey of temporal data mining. *Sadhana* 3: 173-198.
- Mannila, H., Toivonen & H., Verkano, A.I. 1997. Discovery of frequent episodes in event sequences. *Data Mining and Knowledge Discovery*, 1(1): 259-289.