Predictive Sequence Miner in ILP Learning

Carlos Abreu Ferreira[1], João Gama[2], and Vítor Santos Costa[3]

[1] LIAAD-INESC and ISEP - Polytechnic Institute of Porto, Porto, Portugal
[2] LIAAD-INESC and Faculty of Economics - University of Porto, Porto, Portugal
[3] CRACS-INESC and Faculty of Sciences - University of Porto, Porto, Portugal

Abstract. This work presents an optimized version of `XMuSer`, an ILP based framework suitable to explore temporal patterns available in multi-relational databases. `XMuSer`'s main idea consists of exploiting frequent sequence mining, an efficient method to learn temporal patterns in the form of sequences. `XMuSer` framework efficiency is grounded on a new coding methodology for temporal data and on the use of a predictive sequence miner. The frameworks selects and map the most interesting sequential patterns into a new table, the sequence relation. In the last step of our framework, we use an ILP algorithm to learn a classification theory on the enlarged relational database that consists of the original multi-relational database and the new sequence relation.
We evaluate our framework by addressing three classification problems and map each one of three different types of sequential patterns: frequent, closed or maximal. The experiments show that our ILP based framework gains both from the descriptive power of the ILP algorithms and the efficiency of the sequential miners.

1 Introduction

Multi-relational databases are widely used to represent and store data. A multi-relational database is often composed of a *target* table and of a number of *fact* tables. The target table will represent the main objects of interest (say, patients in a medical domain); fact tables will represent the information being accumulated about the entities in the target table (say, medical visits or drug usage in the medical domain). We expect target tables to be relatively stable or to grow slowly over time; in contrast, fact tables may grow quickly. Moreover, quite often the information stored in fact tables is time-based and consists of *sequences* that reflect the evolution of a phenomenon of interest. Referring back to the medical domain, a patient is subject to a sequence of examinations, where a set of measurements, corresponding to results of different analyses, are taken.

In this work, we start from the hypothesis that the evolution of these measurements, as encoded in the fact tables, may hold relevant information for the diagnosis. The problem we address here is *how best to explore such information?* More precisely, we focus on how to learn highly descriptive and accurate decision models given multi-relational data with sequences.

In this paper we present a framework that allows us to explore multi-relational datasets that have different types of temporal data, either sequence data or

time-series data. On the one hand, we can benefit from computationally efficient sequential miners such as cSpade [17] to find the most predictive sequential patterns. On the other hand, we still have access to the original data and can take advantage of the flexibility of Inductive Logic Programming (ILP) to learn in the extended multi-relational dataset. Indeed, we argue that the first step provides a good insight into the search space, and may enable `XMuSer` to perform better than classical ILP based algorithms in large search spaces. We should observe that the sequence miner and ILP learning algorithm are decoupled and we can use a variety of sequential miners to find a constrained set of sequential patterns.

We name our framework `XMuSer`(eXtended MUlti-relational SEquential patteRn knowledge learning). It executes in five steps. First, we encode the multi-relational temporal data into a sequence database. In this new database each example is a heterogeneous sequence that was built regarding both intra-table and inter-table relations within the temporal data. In a second phase, we use a sequence miner to find frequent and class predictive sequences in the sequence database. By class predictive we mean sequential patterns that are frequent in at least one class partition. In a third phase we use the chi-square statistic to sort all sequential patterns. If needed, we introduce a strategy to select a constrained subset of sequential patterns. The fourth phase maps back the top-k most interesting sequential patterns by building a new relation, the *sequence relation*, where each target example is characterized by the presence or absence of each pattern. Finally, we apply an ILP algorithm to learn a theory from the enlarged database, i.e. the union of the original database with the *sequence relation*.

We experimentally evaluate our methodology with two datasets, originally introduced at PKDD Discovery Challenges, addressing three classification problems: *Hepatitis*, that records examination data from patients having hepatitis B or C subtypes; and, *Financial*, that records bank accounts information.

The contributions of our work are therefore: 1. We introduce a framework to explore heterogeneous sources of temporal data, either sequence data or time-series data, stored in multi-relational database using propositional sequence miners; 2. `XMuSer` ILP based framework is highly efficient and gains both from the descriptive power of the ILP algorithms and the efficiency of the sequence miners. We do not use classical aggregation strategies, like time windows, neither neglect valuable logic-relational information; 3. We develop a new methodology to translate any multi-relational temporal database into a sequence database.

In the next Section we present the main ideas of our work and present the related work. In Section 3 we define concepts that will be useful to Section 4, where we discuss in detail our framework. Next, in Section 5, we describe the experimental setup and present and discuss the obtained results. At the end of the paper, we give an overview of this work and present future research directions.

2 Methods and Related Work

In this section we present an overview of related work that inspired us and contributed to the overall `XMuSer` framework.

There exists a wide range of algorithms that can explore sequential data in an efficient way. Agrawal and Srikant introduce the *GSP* algorithm [14], an algorithm that generalizes the original sequential pattern mining problem. *GSP* uses a candidate-generation strategy to find all frequent sequences, and uses a lattice to generate all candidate sequences. Different from GSP, that uses horizontal layout format, SPADE [18] algorithm uses vertical layout format where each sequence in the lattice is associated with the *idlist*. This *idlist* is a list of all example sequences containing the candidate sequence, the list contains a set of pairs where each pair consists of both a sequence id(*sid*) and an event identifier (usually the event time). To search for frequent patterns SPADE uses candidate-generation strategy. To compute the support of each candidate pattern of level l, the *idlist*'s of sequences from level $l-1$ are joined using a *temporal join*. The support of each candidate pattern is the number of distinct *sid* in the candidate *idlist*. Like GSP, SPADE uses Apriori property to prune the search space. In real problems these algorithms usually find a huge number of uninteresting sequence patterns. To solve this issue of sequential miners, algorithms such as *CloSpan* [16], that returns a set of closed sequential patterns, and *SPIRIT* [7] that constrains the search space using regular expressions were developed. cSPADE [17] extends SPADE algorithm to find a constrained set of sequential patterns. By using cSPADE we can find, among other, sequential patterns predictive of one or more classes. The author observes that by using highly predictive sequences as input to a propositional classifier we can get accuracy gains that range from 10% to 50%.

A different approach, that is known to be successful, is to use post-processors, *filters*, to select interesting patterns [6]. Some use sequential ad-hoc selection, that are model unrelated, whereas others use wrapper filters, that select features based on the induced models.

Algorithms that use Inductive Logic Programming (ILP) were the first ones to explore successfully the richness of multi-relational data. The standard ILP algorithm uses a greedy covering approach to induce a set of rules, often represented as Prolog clauses, that together form a theory. There are different approaches to generate the rule set. Algorithms like FOIL [13], Progol [9], and Aleph [15], use a top-down refinement approach whereas others like GOLEM [10] use a bottom-up strategy. ILP approaches have an enormous representational power but are often criticized for lacking scalability [1]: ILP algorithms may not be very effective for the large search spaces induced by sequence databases.

One approach to solve the above mentioned issue is to use propositionalization with ILP [19]. The idea is to augment the descriptive power of the target table by projecting clauses (new attributes) on the target table.

Even with recent progress on scalability there exists multi-relational data which remains almost impossible to explore effectively by using only ILP based approaches. As an example, intra-table and inter-table temporal patterns remain hard to explore. One approach, followed by *WARMR* [4] is to use aggregation methodologies, unfortunately losing relevant time information. In View Learning [3] we can define and use alternative *views* of the database, i.e., we can

define new fields or tables. Such new fields or tables can also be highly useful in learning, but still require searching a very large search space.

Other ILP based approaches develop specific techniques aimed at exploring the space and time information available in multi-relational datasets. The works of [8, 5] introduce special purpose formalisms to find first-order sequential patterns in multi-relational datasets but, by using ILP based search, they suffer from traditional ILP limitations. To explore large spaces they must constrain the search space or use heuristics, otherwise the problems will be intractable. Thus, they may fail to find interesting patterns.

3 Preliminaries

We will start by defining the sequence mining problem and then we introduce some concepts.

Let $I = \{i_1, i_2, \ldots, i_n\}$ be a set of items and e an *event* such that $e \subseteq I$. A *sequence* is an ordered list of events $e_1 e_2 \ldots e_m$ where each $e_i \subseteq I$. Given two sequences $\alpha = a_1 a_2 \ldots a_k$ and $\beta = b_1 b_2 \ldots b_t$, the sequence α is called a *subsequence* of β if there exist integers $1 \leq j_1 < j_2 < \ldots < j_k \leq t$ such that $a_1 \subseteq b_{j_1}, a_2 \subseteq b_{j_2}, \ldots, a_k \subseteq b_{j_k}$. A *sequence database* is a set of tuples (sid, α) where *sid* is the sequence identification and α is a sequence. The *count* of a sequence α in a sequence database D, denoted $count(\alpha, D)$, is the number of sequences in D containing the α subsequence. The *support* of a sequence α is the ratio between $count(\alpha, D)$ and the number of sequences in D. We denote support of a sequence by $support(\alpha, D)$. Given a sequence database D and a minimum support value λ, the problem of sequence mining is to find all subsequences in D having at least a support value equal to λ. Each one of the obtained sequences is also known as a *frequent sequence*.

We can also use sequence miners to find a constrained set of sequential patterns. A frequent sequence α is a *closed sequential pattern* if there exists no proper supersequence β having the same support as α. A frequent sequence α is a *maximal sequential pattern* if it is not a subsequence of any other frequent sequence. The set of maximal sequential patterns is a subset of all closed sequential patterns.

These two types of sequences can be obtained using two strategies. One is to develop specialized algorithms that find a specific type of patterns, for instance CloSpan [16] can find closed sequential patterns. The other way is to use a post-processing strategy to select a subset of patterns among all frequent patterns.

For instance, to find the set of *closed sequential patterns* we can use the following naive strategy: 1. Find all frequent sequences; 2. Sort descending all sequential patterns according to each one support value and, as a second criterion, sort using the length of each pattern; 3. Top-down searches the sorted set of sequential patterns and, for each sequential pattern, eliminate all subsequences having equal support value. In the end, the remaining patterns are the set of closed sequential patterns.

To find all maximal sequential patterns we can use the following naive strategy: 1. Find all frequent sequences; 2. Sort descending all sequential patterns according to their length; 3. Top-down searches the sorted set of sequential patterns and, for each sequential pattern, eliminate all its subsequences that appear after the given sequence. In the end, the remaining patterns are the set of maximal sequential patterns.

When addressing a classification problem we can explore predictive sequences, sequential patterns that are frequent in at least one class partition. In such problems a sequence miner takes as input a sequence database where each tuple has an associated class label, $(sid, \alpha, class)$.

Such predictive patterns are valuable but some can have low discriminative power. To select high discriminative sequential patterns we can use several techniques from contrast set mining, emerging pattern mining or subgroup discovery [11]. For instance, for each sequential pattern, we can compute the chi-square statistic from a contingency table or compute the support difference across class partitions. Using one of these strategies we can select valuable patterns from the sequence database that could be useful at theory learning time [19].

To include such valuable information in the final classification model we can map the most interesting sequential patterns [19]. Thus, we introduce the concept of Sequence Relation and Enlarged Database.

Definition 1. (Sequence Relation and Enlarged Database) *Consider a multi–relational database* $\mathbf{r}$*, a sequence database* D *coded from* $\mathbf{r}$*, where each sequence id equals the primary key of* $\mathbf{r}$*'s target table. Also consider the set of sequential patterns* S *obtained from solving the sequential mining problem. We define the sequence relation,* $\mathbf{r}_{sr}$*, to be the set of tuples* $(sid, \alpha_1^B, \alpha_2^B, \ldots, \alpha_n^B)$ *where sid is the sequence id in* D *and* α_i^B *is a binary attribute whose value is obtained according to the projection of the sequential pattern* α_i *in the sequence database. The enlarged database is the database resulting from the union of the multi-relational database and the sequence relation. Formally we define the enlarged database as being* $\mathbf{E_r} = \mathbf{r} \cup \mathbf{r_{sr}}$.

4 Algorithm Description

In this section we present `XMuSer`, a framework developed to explore multi-relational temporal information, mainly heterogeneous sequential data. The main idea of the algorithm is to explore work developed in the sequential pattern mining field to include temporal information in the ILP learning process.

The framework has five main steps. In the first phase, if needed, we code the temporal data into a sequence database. In the second phase, we run a sequence miner to find all predictive sequential patterns. In the third phase, we sort the predictive sequential patterns using the chi-square statistic. In the fourth phase, we select the top-k most interesting sequential patterns and, for each example in the target table, we build a relation where the example is characterized by presence or absence of the k most interesting sequential patterns. Last, we learn

Algorithm 1: Framework pseudo-code

input : a multi-relational database $\mathbf{r}$; two thresholds, λ, the sequence miner support value, and k, the number of interesting patterns to map
output: a first-order classifier model

1 **Sequence Coding**
$\mathbf{s} \leftarrow$ `SequenceCoding` $(\mathbf{r})$
2 **Find Frequent Sequential Patterns**
$\mathbf{fs}_{patterns} \leftarrow$ `SequenceMiner` $(\mathbf{s}, \lambda)$
3 **Sort patterns according to some criterion**
$\mathbf{fs}_{rank} \leftarrow$ `rank` $(\mathbf{fs}_{patterns})$
4 **Mapping top-k patterns**
$\mathbf{r}_{sr} \leftarrow$ `Mapping(`$r_{targetExamples}$,$\boldsymbol{fs}_{rank}$,k`)`
$\mathbf{E_r} \leftarrow \mathbf{r} \cup \mathbf{r}_{sr}$
5 **Learn a Theory with an ILP algorithm**
`ILP Algorithm(`$\mathbf{E_r}$`)`

a theory on the enlarged database, where enlarged database is the union of the original database with the new sequence relation.

Algorithm 1 presents the pseudo-code for `XMuSer`. Next, we explain each one of the major components in self-contained subsections. Throughout, we follow an illustrative example, a classification problem, presented in Figure 1. This example is inspired on the relational Hepatitis dataset. The example has three tables registering the follow-up of two patients. One of the tables is the target table, named *Patient*, where each record describes each patient, identified by a masked ID, and registers the class of each patient. The other two tables are fact tables registering temporal blood analysis and urinalysis examinations. The *Blood Analysis* table registers, for each patient, the examination date and the patient's RBC and WBC parameters. Table *Urinalysis* registers for each patient the value of three parameters: *alb*, *plt* and *ttp*.

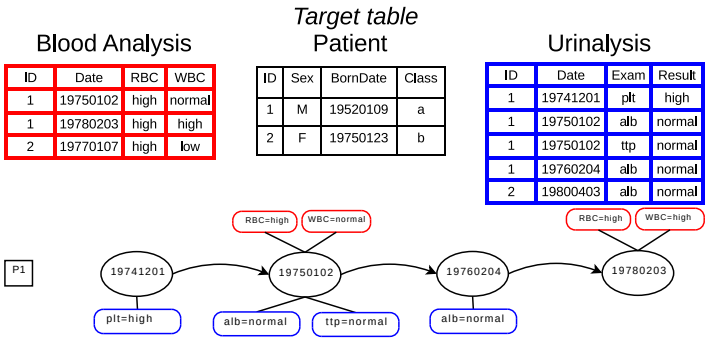

Fig. 1. Database Relations (top) and Patient One Event Sequence (bottom)

Table 1. Sequence Database

ID	Sequence	Class
1	(9) (3 5 7 8) (7) (3 6)	a
2	(3 4) (7)	b
3	(1 5) (4)	b
4	(3 5) (7)	a
5	(5 8) (7)	a
6	(4)	b

Data Coding Here we introduce the strategy that converts the multi-relational temporal data into an amenable sequence database. The idea is for each example in the multi-relational target table to find all associated relations that have temporal records. Next, we sort all such records by time order. In this way we obtain a chronological sequence of multiple events for each example. In Figure 1 we present one example event sequence. In this case, the sequence includes a sequence of blood and urine analysis.

Next, and following time-order, we build an attribute-value sequence for each example. As an example, for patient one we get $(plt = high)(RBC = high, WBC = normal, alb = normal, ttp = normal)(alb = normal)(RBC = high, WBC = high)$. In this new sequence each item corresponds to all records registered at a given date/time. Then, we define a one-to-one coding map $f : Attributes \times Values \longrightarrow \mathbb{N}$. This mapping associates a *unique* number to each attribute-value pair. In the example, we use the map to code the attribute-value sequence into an integer number sequence. The definition of this map is done according to the type of attributes in each database relation. For each discrete attribute we find the range of attribute values and map each attribute-value pair onto a unique integer value. This map is a three element tuple having the attribute name, the attribute value and the unique integer number that identifies the attribute-value. When dealing with continuous attributes, first we apply a discretization strategy and then proceed as for discrete attributes. Discretization will be problem and domain specific.

Table 1 shows the resulting sequence database: each sequence tuple corresponds to an example in the target table and each subsequence corresponds to all one-time events. Following the simple example, and defining the one-to-one map $f(rbc, low) = 1$, $f(rbc, normal) = 2$, $f(rbc, high) = 3$, $f(wbc, low) = 4$, $f(wbc, normal) = 5$, $f(wbc, high) = 6$, $f(alb, normal) = 7$, $f(ttp, normal) = 8$, $f(plt, high) = 9$, patient one sequence of events is coded into the sequence (9) (3 5 7 8) (7) (3 6).

After this preprocessing stage we get a sequence relation suitable to be used by a sequence miner. In Table 1 we present a sequence database registering the coded sequence of patients one and two and four other patients, that were not present in the example database presented in Figure 1, but will be useful to explain some steps of our algorithm. In the third column we show the labels of each sequence, we have two classes, class **a** and class **b**.

Table 2. Predictive Sequential Patterns found

	Support Value			
Predictive Sequential Patterns	Overall	Class **a**	Class **b**	Chi-square
(3)	3	2	1	0.667
(4)	3	0	3	6.000
(5)	4	3	1	3.000
(7)	4	3	1	3.000
(3)(7)	3	2	1	0.667
(5)(7)	3	3	0	6.000
(3 5)	2	2	0	3.000

Finding Frequent Sequential Patterns To explore the sequence database coded in the previous step we can fit any sequence miner.

Our first step is to run a sequence miner on the sequence database to find predictive frequent sequences, the ones having, in at least one class partition, a support value equal or higher than a user defined threshold. Following the illustrative example, if we set the support value to 0.5 and run cSPADE algorithm to find sequential patterns available in the sequence database, Table 1, we get the sequential patterns presented in Table 2. The original cSPADE algorithm outputs the support of each pattern. cSPADE outputs the overall support of each pattern and the support of each pattern in each class partition. Moreover, we slightly modify cSPADE to compute a measure of interest for each pattern. Here, we set cSPADE to output the chi-square statistic for each pattern.

Considering the nature of the sequence miner, the type and the dimension of problem being addressed we usually get a large number of sequential patterns and some of these patterns can be redundant or uninteresting. To retain highly interesting patterns and class correlated patterns we use the following strategy.

Sort Sequential Patterns To select the most interesting patterns to build the final classification model, we sort all the sequential patterns available in Table 2 using a metric. In this example, we sort the patterns according to the chi-square statistic value. The k most interesting patterns (the top ones) will be used to build the final classification model.

Mapping Back Interesting Sequences In order to best explain our procedure we present Table 3, the sequence relation that was obtained by applying

Table 3. Sequence Relation

ID	S_1	S_2	S_3
1	true	false	true
2	false	true	false

our mapping procedure on the example data. This table has four attributes, the example ID and the three most interesting sequences.

We develop a mapping strategy that maps each one of the selected sequential patterns into a Boolean attribute and, portray this mapping as a new table. This new table, named *sequence relation*, has an entry for each example on the target table and has $k+1$ attributes: the top-k most interesting features and the example ID, usually the target table primary key. To compute the value of each attribute we use the information coded in the sequence database and subsequence definition. We apply the following rule: If the sequence associated with the new attribute is a subsequence of the example sequence at the sequence database, the new attribute takes value *true*. Otherwise the attribute takes value *false*. Using this procedure we get a new table that represents the most interesting temporal data available in the multi-relational database.

Learning a Theory In this last step we learn a theory. We take all the primitive tables and add the new sequence relation as input to an ILP algorithm, such as Aleph, to learn a set of clauses. These clauses can therefore use the primitive tables and the new one, that encodes the temporal information.

One example of an illustrative clause we can find is:
patient_info(A,B,C,a) :- blood_analysis(A,D,high,normal), urinalysis(A,E,plt,high), sequence_relation(A,true,F,G).
This clause has the predicate *patient_info* at the head, and a call to the predicate *blood_analysis*, the predicate *urinalysis* and the predicate associated with the sequence relation, predicate *sequence_relation*, as the clause body. The clause explains (or covers) patient number one in the database, a patient from class **a**.

5 Experimental evaluation

When defining the experimental configuration to evaluate our framework we were constrained by the representational complexity and the amount of data. We use the YAP Prolog compiler [2] to implement three tasks: converting the temporal data presented in the multi-relational database into a sequence database, that can be fed to the cSPADE algorithm format; map the most interesting sequential patterns on the sequence relation; and run the Aleph algorithm to learn a theory.

Using these tools, we define two configurations of the input parameters and run `XMuSer` to solve three classification problems available in two datasets: the Hepatitis Subtype, the Hepatitis Fibrosis degree and the prediction of successful loans (the Financial problem). The Hepatitis Fibrosis degree problem is a multi-class problem that we cast into a binary problem.

We test two different sequence miner support values (the λ parameter), 90% and 80%, and study the sensibility of our framework to the number of sequential patterns we map in the fourth phase (the k parameter). In this last study, we map a number of patterns ranging between one and fifteen patterns. For comparison purposes we also run the stand-alone Aleph algorithm to solve each one of the four problems. Furthermore, besides presenting `XMuSer` results using all

sequential patterns found by cSPADE, we present results using two subsets of sequential patterns, the closed set and the maximal set. The closed and maximal sequences are obtained using a post-processing strategy and the naive strategy described in Section 3. This way we define three instances of `XMuSer`.

We evaluate our framework using a ten-fold cross-validation procedure and compute: the mean number of patterns found after each step of `XMuSer`, the generalization accuracy mean and standard deviation of `XMuSer` and the mean time spent to complete each step of `XMuSer`. Using this same validation procedure, we also compute: the mean number of rules learned by the Aleph algorithm, both as a component of `XMuSer` and as a stand-alone algorithm. We also compute the generalization accuracy and the time spent by the stand-alone Aleph algorithm. Concerning the generalization accuracy, we also compute the Wilcoxon hypothesis test p-value to measure how significantly our algorithm differs from the reference algorithm, the stand-alone Aleph algorithm. We set the significance level to 0.05.

We present the results that we obtained by introducing a strategy to select the optimum number of sequential patterns to map. To select the optimum number of patterns we use only 40% of the data and run a 4-cv experiment. The optimum number of patterns is the point having maximum mean generalization accuracy.

Concerning the comparison, we use the same background knowledge and bias when running the Aleph algorithm, either using Aleph as a component of the `XMuSer` framework or by running it as stand-alone reference algorithm. The major difference is that when running the Aleph as a component of `XMuSer` we allow extra refinements by introducing the sequence relation. As our goal is to evaluate the contribution of our novel technique, our goal is more to proof the contribution of our framework and less to search for the best overall results. Thus, we run Aleph algorithm using a relatively simple bias, relying on the predefined tables in the database.

Datasets and Tasks We present below the two datasets that we used to evaluate our ILP based classifier: Hepatitis and Financial datasets. Both these datasets are available to download at the PKDD challenge web page[4].

The Hepatitis dataset consists of seven tables registering a long term, from 1982 to 1990, monitoring of 771 patients having hepatitis B or C. One table provides personal data about patients. The other tables record blood and urinalysis examinations. We address two classification problems. Our first task is to discriminate between patients having B and C hepatitis. The second task is to determine the degree of liver fibrosis. Following previous work [12], we study fibrosis degree 2 and 3 against fibrosis degree 4. We perform limited feature selection, based on the dataset description [12]. We select GOT, GPT, TTT, ZTT, T-CHO, CHE, ALB, TP, T-BIL, D-BIL, I-BIL, ICG-15, PLT, WBC and HGB features. As these features are numerical, and in fact take a wide range of values, we discretized each feature according to medical knowledge. We use

[4] `http://lisp.vse.cz/challenge/CURRENT/`

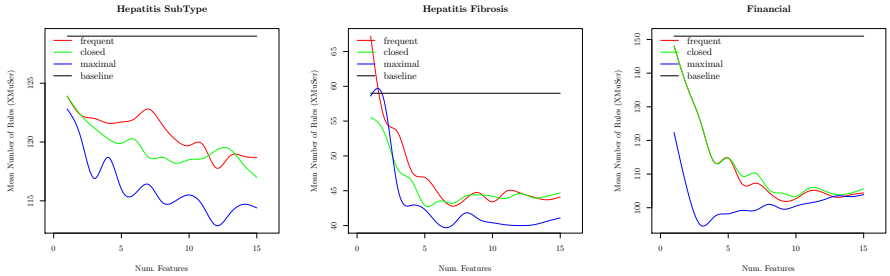

Fig. 2. Mean number of rules found by running ALEPH(the last component of `XMuSer`) with support value λ set to 0.8. We also present the mean number of rules that we obtain by running the stand-alone Aleph algorithm.

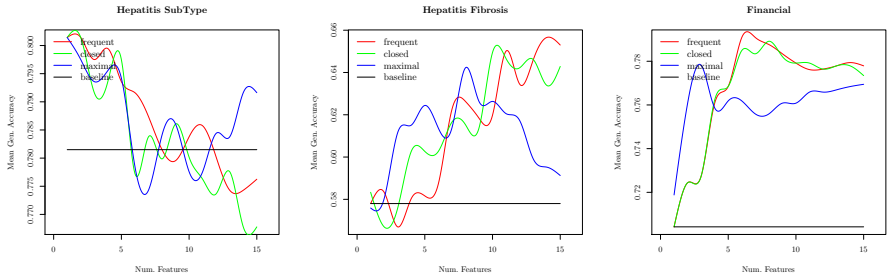

Fig. 3. `XMuSer` mean generalization accuracy by setting the support value, λ, to 0.8. We also present the mean generalization accuracy when we run the stand-alone Aleph algorithm.

three bin values, low, normal and high. For the sub-type problem we end with 206 hepatitis-B patients and 297 hepatitis-C patients. For the Fibrosis problem, we have 209 patients of {2,3}-class and 67 of the {4}-class.

The Financial dataset includes eight tables storing data about clients of a bank. A number of tables store static information on accounts, clients and regional demographics. The remaining tables register information concerning credit card types, payments, transactions, and loans for each account. The sequence database relies on the *Balance* attribute only. We discretized this attribute using the inequality $|x - \mu| < 2\sigma$, where μ is the attribute mean value, σ is the standard deviation and x is the attribute value that we map into one of three bin values: low, normal and high. Our target is predicting successful loans: 606 loans were classified as successful and 76 unsuccessful.

Results In this section we present the results that we obtained when running `XMuSer` and the stand-alone Aleph algorithm. In Figure 2 we present the mean

number of rules found by the last component of XMuSer, the Aleph algorithm. In figure 3 we present the mean generalization accuracy of XMuSer. We do not show the results when we set $\lambda = 0.9$ as we do not observe significant changes. Moreover, for each dimension of analysis, we plot the mean number of rules for the frequent, the closed and the maximal version of XMuSer framework. We also present the results that we obtain when running the baseline, the stand-alone Aleph algorithm.

In Tables 4, 5 and 6 we present the mean number of patterns, the mean generalization accuracy and the mean run-time, respectively, that we obtain by using an optimum number of sequential patterns to map. Remember that our selection strategy uses a sample of each dataset. The optimum number of patterns to map is presented in Table 4, columns six, seven and eight.

Analysis First of all, we must say that most of the works that address these same problems, use preprocessing procedures, like removing some examples to balance the datasets, that makes almost impossible to make a fair comparison against other works.

We present results only for two λ values, 90% and 80%. For lambda values higher than 90% we do not observe any significant change in comparison against the 90% value. Regarding the values lower than 80%. First, we are limited by the available computational power. With the current resources, if we set lambda parameter to values lower than 80% we are unable to run XMuSer to solve all problems. Second, in the datasets that we were able to run XMuSer we do not identify significant changes in the results. In all three classification problems, XMuSer obtained better or equal results in comparison against the stand-alone Aleph algorithm. Moreover, the best accuracy gains were obtained in Hepatitis Fibrosis and in the Financial problem. If we run XMuSer by setting k to the optimum number of patterns to map, we can get significant wins for all problems.

In all problems that we address we can get a small improvement if we use a higher number of patterns but at the cost of getting large and less interpretable theories.

The classifier models that we learn were obtained using the same ILP bias but produce better or equal results than the reference algorithm. The results obtained in the Hepatitis dataset are among the best ones that we could find in related

Table 4. Mean number of patterns in each step of XMuSer (using optimized number of patterns) and of the stand-alone Aleph algorithm

		XMuSer (With Aleph)									Stand-alone Aleph (Rules)
		Seq. Miner			Map			Aleph(Rules)			
	λ	Freq.	Clo.	Max.	Freq.	Clo.	Max.	Freq.	Clo.	Max.	
Hepatitis Subtype	0.9	14	14	10	1	1	1	128	128	128	129
	0.8	2444	1824	732	3	1	1	122	124	122	
Hepatitis Fibrosis	0.9	21	21	16	4	1	1	47	53	55	59
	0.8	2325	1817	709.4	13	10	4	44	44	43	
Financial	0.9	140	126	98	4	4	12	141	141	142	152
	0.8	6992	5633	920	7	7	14	107	110	103	

Table 5. Mean Generalization Accuracy: `XMuSer` (using optimized number of patterns) against Stand-alone Aleph

		`XMuSer` (With Aleph)			Stand-alone Aleph	Wilcoxon p-value		
	λ	freq.	clo.	max.		freq.	clo.	max.
Hepatitis Subtype	0.9	0.780(0.11)	0.778(0.11)	0.778(0.11)	0.785(0.11)	0.622	0.106	0.106
	0.8	0.799(0.10)	0.801(0.10)	0.801(0.11)		0.049	0.013	0.013
Hepatitis Fibrosis	0.9	0.642(0.05)	0.632(0.05)	0.628(0.07)	0.578(0.09)	0.032	0.138	0.166
	0.8	0.640(0.07)	0.649(0.05)	0.615(0.09)		0.126	0.232	0.322
Financial	0.9	0.749(0.05)	0.749(0.05)	0.743(0.04)	0.704(0.07)	0.123	0.123	0.075
	0.8	0.791(0.03)	0.783(0.03)	0.768(0.02)		0.009	0.009	0.066

Table 6. Mean Run-Time, in seconds, of each `XMuSer` phase (using optimized number of patterns) and the stand-alone Aleph algorithm

		`XMuSer` (With Aleph)									Stand-alone Aleph (Rules)
		Seq. Miner	Naive Selection		Map			Aleph(Rules)			
	λ	Freq.	Clo.	Max.	Freq.	Clo.	Max.	Freq.	Clo.	Max.	
Hepatitis Subtype	0.9	0.1	0	0	1.5	1.6	1.6	6.1	6.1	5.9	6
	0.8	1.8	0.08	1.95	1.8	1.4	1.45	9.7	6.4	5.9	
Hepatitis Fibrosis	0.9	0	0	0	1.1	1	1	5.4	4.1	4.3	5
	0.8	1.2	0.15	1.43	1	1.4	2.6	11.6	10	6.5	
Financial	0.9	0.4	0	0	0.6	0.5	0.7	3.3	3.1	14.4	2
	0.8	12.3	1.4	28.04	0.8	2.6	29.8	5.3	4.5	12.2	

work that addresses this problem, and indeed outperform our previous work where the final step is a propositional classifier [6]. This is especially relevant since we did not customize the search bias to obtain the most accurate results.

One important point that proves that we are in the right direction is the relation between the accuracy of the obtained classification models and the number of temporal data attributes that we use to learn the final classification theory. When addressing the three problems where we get a significant win, the final theory always include sequential patterns mapped in the sequence relation. The two problems where we get the higher accuracy wins are the ones where `XMuSer` generates more compact theories. These compact theories include the sequence relation predicate, i.e., include all, or some, sequential patterns coded in the sequence relation. Moreover, by setting Aleph to use predicates like $sequence\_relation(+id, \#a, \#a, \#a)$ to construct each rule we can get interesting patterns that represent a conjunction of positive and negative sequential patterns. Such clauses, that can include negation of some or several sequential patterns, can be valuable in many domains.

As expected, `XMuSer` execution time is greater than the stand-alone Aleph algorithm. The execution time heavily depends on the two input parameters, the λ and k values. Moreover, when we set λ parameter to low values, the time need to select closed or maximal sequential patterns, especially the last ones increases. If we run `XMuSer` by setting λ input parameter to low values we get a huge number of sequential patterns and thus we get a run-time overhead. This is a typical behavior of sequence miners, the lower the support the higher

the execution time and the number of patterns found. In respect to the post-processing strategies that we use to select the closed sequential patterns and the maximal sequential patterns, we can say that the time needed to select the maximal sequential patterns is higher than the time needed to select the closed sequential patterns. This is especially clear when we run the sequence miner with 0.8 support value.

Furthermore, if we analyze theoretically the run-time of the last step of `XMuSer`, we can say that by adding the sequence relation to the search bias we will have to explore a 2^k larger search space, where k is the number of patterns/attributes in the sequential relation. In our experiments, when comparing the mean run-time of last step of our framework against the stand-alone Aleph algorithm, the time spent by the last component of our framework is almost the same as the run-time of the stand-alone Aleph algorithm. If we stay on low k values we can effectively explore the temporal data using our framework.

6 Conclusions and Future Work

In this work we have presented `XMuSer` a framework whose efficiency is grounded in sequence data representation and in the strengths of sequential miners. Our approach is general and can be used in conjunction with any classical ILP algorithm.

To design our five step architecture we introduce some new methodologies that are at the core of `XMuSer`. We developed sequence coding, a technique to code the temporal data available in the original database into a sequence database. Regarding the number of findings of the sequential miner and the need to obtain class correlated sequential patterns among these findings, we introduce a filter that selects the top class related sequential patterns. We use the chi-square statistical to sort the sequential patterns and map the top most interesting ones. We introduce the sequence relation, a relation that encodes the time information of each example, the sequential patterns found by the sequential miner. Last, we define the enlarged database, a database that is the union of the original database and the sequence relation, and induce a first-order theory on the enlarged database. We believe that this methodology ensures that we can explore the valuable logic-relational information available in multi-relational datasets, especially temporal patterns, using an ILP learner, and this is confirmed by our empirical evaluation of our framework.

In the future, we will develop specialized sequential miners that can find first-order closed or maximal predictive sequential patterns in a multi-relational classification framework. Moreover, we will study other metrics to sort the sequential patterns aiming to obtain highly interesting patterns. By doing this we can go further ahead and explore reacher temporal data available in relational datasets that is impossible to explore if we were grounded only in ILP learning.

Acknowledgments: This work was supported by the Portuguese Foundation for Science and Technology (FCT) under the projects KDUS (PTDC/EIA-EIA/098355/2008) and HORUS (PTDC/EIA-EIA/100897/2008).

References

1. Blockeel, H., Sebag, M.: Scalability and efficiency in multi-relational data mining. SIGKDD Explorations 5(1), 17–30 (2003)
2. Costa, V.S.: The life of a logic programming system. In: Logic Programming, 24th International Conference, ICLP 2008, LNCS 5366. vol. 5366, pp. 1–6. Springer, Udine, Italy (2008)
3. Davis, J., Burnside, E., Ramakrishnan, R., Costa, V., Shavlik, J.: View learning for statistical relational learning: With an application to mammography. In: Proceeding of the 19th International Joint Conference on Artificial Intelligence. pp. 677–683. Professional Book Center, Edinburgh, Scotland, UK (2005)
4. Dehaspe, L., Toivonen, H.: Discovery of frequent DATALOG patterns. Data Mining and Knowledge Discovery 3(1), 7–36 (1999)
5. Esposito, F., Di Mauro, N., Basile, T.M.A., Ferilli, S.: Multi-dimensional relational sequence mining. Fundamenta Informaticae 89(1), 23–43 (2009)
6. Ferreira, C.A., Gama, J., Costa, V.S.: Sequential pattern mining in multi-relational datasets. In: Proceedings of the 13th Conference of the Spanish Association for Artificial Intelligence, LNCS 5988. pp. 121–130. Springer, Seville, Spain (2010)
7. Garofalakis, M., Rastogi, R., Shim, K.: Mining sequential patterns with regular expression constraints. IEEE Transactions on Knowledge and Data Engineering 14(3), 530–552 (2002)
8. Lee, S.D., Raedt, L.D.: Constraint based mining of first order sequences in seqlog. In: Database Support for Data Mining Applications: Discovering Knowledge with Inductive Queries, LNCS 2682. pp. 155–176. Springer (2004)
9. Muggleton, S.: Inverse entailment and Progol. New Generation Computing, Special issue on Inductive Logic Programming 13(3&4), 245–286 (1995)
10. Muggleton, S., Feng, C.: Efficient induction of logic programs. In: Algorithmic Learning Theory, First International Workshop. pp. 368–381. Springer/Ohmsha, Tokyo, Japan (1990)
11. Novak, P.K., Lavrač, N., Webb, G.I.: Supervised descriptive rule discovery: A unifying survey of contrast set, emerging pattern and subgroup mining. Journal Machine Learning Research 10, 377–403 (June 2009)
12. Ohara, K., Yoshida, T., Geamsakul, W., Motoda, H., Washio, T., Yokoi, H., Takabayashi, K.: Analysis of Hepatitis Dataset by Decision Tree Graph-Based Induction (2004)
13. Quinlan, J.R., Cameron-Jones, R.M.: Induction of logic programs: Foil and related systems. New Generation Computing 13, 287–312 (1995)
14. Srikant, R., Agrawal, R.: Mining Sequential Patterns: Generalizations and Performance Improvements. In: 5th International Conference on Extending Database Technology, LNCS 1057. pp. 3–17. Springer, Avignon, France (1996)
15. Srinivasan, A.: The Aleph Manual (2003), `http://www.comlab.ox.ac.uk/activities/machinelearning/Aleph/aleph.html`
16. Yan, X., Han, J., Afshar, R.: Clospan: Mining closed sequential patterns in large datasets. In: Proceedings of the Third SIAM International Conference on Data Mining. pp. 166–177. SIAM, San Francisco, CA, USA (2003)
17. Zaki, M.J.: Sequence mining in categorical domains: Incorporating constraints. In: CIKM. pp. 422–429 (2000)
18. Zaki, M.J.: Spade: An efficient algorithm for mining frequent sequences. Machine Learning 1(42), 31–60 (2001)
19. Zelezny, F., Lavrac, N.: Propositionalization-Based Relational Subgroup Discovery with RSD. Machine Learning 62(1-2), 33–63 (2006)