Fundamenta Informaticae 89 (2008) 23–43
IOS Press

Multi-Dimensional Relational Sequence Mining

Floriana Esposito, Nicola Di Mauro, Teresa M.A. Basile and Stefano Ferilli
Università degli Studi di Bari, Dipartimento di Informatica, 70125 Bari, Italy
{*esposito,ndm,basile,ferilli*}*@di.uniba.it*

Abstract. The issue addressed in this paper concerns the discovery of frequent multi-dimensional patterns from relational sequences. The great variety of applications of sequential pattern mining, such as user profiling, medicine, local weather forecast and bioinformatics, makes this problem one of the central topics in data mining. Nevertheless, sequential information may concern data on multiple dimensions and, hence, the mining of sequential patterns from multi-dimensional information results very important. In a multi-dimensional sequence each event depends on more than one dimension, such as in spatio-temporal sequences where an event may be spatially or temporally related to other events. In literature, the multi-relational data mining approach has been successfully applied to knowledge discovery from complex data. However, there exists no contribution to manage the general case of multi-dimensional data in which, for example, spatial and temporal information may co-exist. This work takes into account the possibility to mine complex patterns, expressed in a first-order language, in which events may occur along different dimensions. Specifically, multi-dimensional patterns are defined as a set of atomic first-order formulae in which events are explicitly represented by a variable and the relations between events are represented by a set of dimensional predicates. A complete framework and an *Inductive Logic Programming* algorithm to tackle this problem are presented along with some experiments on artificial and real multi-dimensional sequences proving its effectiveness.

Keywords: Multi-relational sequence mining, Inductive Logic Programming, Sequence analysis

1. Introduction

The rapid growth of the amount of data stored in large databases has lead to an increasing interest in the data mining research area and, in particular, towards methods to discover hidden structured patterns in large databases. The sequences are the simplest form of structured patterns and different methodologies have been proposed to face the problem of sequential pattern mining, firstly introduced by R. Agrawal and R. Srikant in [2], with the aim of capturing the existent maximal frequent sequences in a given

Address for correspondence: Università degli Studi di Bari, Dipartimento di Informatica, 70125 Bari, Italy database. The issue of discovering all frequent sequences of itemsets in a dataset is crucial when the data to be mined have some sequential nature like events in the case of temporal information. Furthermore, some real world domains such as user profiling, medicine, local weather forecast and bioinformatics show an inherent propension to be modelled by means of sequences of events/objects related to each other. This great variety of applications of sequential pattern mining makes this problem one of the central topics in data mining as showed by the research efforts produced in recent years [1, 44, 15, 34, 35] .

However, some environments involve very complex components and features. Thus, the classical existing data mining approaches, that look for patterns in a single data table, have been extended to the multi-relational data mining approaches that look for patterns involving multiple tables (relations) from a relational database. This has led to the exploitation of a more powerful knowledge representation formalism as first-order logic. Some works facing the problem of knowledge discovery from spatial and temporal data in multi-relational data mining research area are present in literature [31, 36, 11, 38, 28]. Nevertheless, there exists no contribution presenting a framework to manage the general case of multi relational data in which, for example, spatial and temporal information may co-exist.

On the other hand, it is worth to note that sequential information might concern data on multiple dimensions and, hence, the mining of sequential patterns from multi-dimensional information turns out to be very important. An attempt to propose a (two-dimensional) knowledge representation formalism to represent spatio-temporal information based on multi-dimensional modal logics is proposed in [6], while the first work presenting algorithms to mine multi-dimensional patterns has been presented in 2001 by Pinto et al. [35]. However, all the works in multi-dimensional data mining have been restricted to the propositional case, not involving a first-order representation formalism.

In this paper we provide an Inductive Logic Programming (ILP) [32] algorithm for discovering *first-order* (DATALOG) *maximal frequent patterns* in *multi-dimensional* relational sequences. Multi-dimensional patterns are defined as a set of atomic first-order formulae in which events are explicitly represented by a variable and the relations between events are represented by a set of dimensional predicates.

Although encoding temporal predicates in ILP is very simple, making a system able to understand and use their semantic is crucial for efficiency. Some recent works on mining logical patterns [18, 25, 29, 4] take into account temporal sequences (i.e., 1-dimensional sequences) by using a purposely defined logical temporal formalism. Instead, this work proposes a dedicated framework which incorporates a specific language bias for multi-dimensional data, expressed in a first-order logic, in order to rise a faster execution and a smaller search space. The first-order logic representation gives us the possibility to encode temporal, spatial and other dimensional features without requiring to discriminate between them. Furthermore, it is possible to represent any other domain relations and let them co-exist with other dimensional ones.

An interesting application of multi-dimensional logical pattern mining is modelling. In particular, considering the user profiling domain and more specifically the user modelling task, an intelligent framework should take into account non only a person's attitude and preferences but also its behaviour and regularities when *moving* in a prespecified context. This might be very useful in improving the interaction between the user and the context itself, in order for the latter to adapt more easily and straightforwardly the functionality that it implements to the former. Indeed, some domains have a stronger need for user models to tune the interface characteristics and behaviours: among the most classical ones, Computer Aided Instruction and Dialog Systems.

A user model can describe the user at different levels of granularity and complexity, depending on the amount of resources available and on the specific task it is intended for. Building user models, however, is a very difficult task, because very often a person's behaviour and preferences vary in time and according to different environments, situations and objectives [20, 41]. Furthermore, the task is made harder when context parameters are taken into account. The classical approaches to context modelling consider low level informations coming from sensors. Some of these works are Synapse [39] system, that, by using Hidden Markov Model, learns user habits exploiting a chronological order of the contexts in order to predict and provide the more relevant functionalities in a given context. SenSay [40] exploits Bayesan Networks to model a context-aware cellular phone that adapts its functionalities (e.g., changing ring or type tone, turning down an incoming call). In [37] a neural network is trained with information provided by sensors concerning the speed of moving of the user and the environment in which the user is, to learn models for an intelligent tourists' guide that, for example, is quiet when the user runs, presents a graphical interface when the user is still or is sitting. Similarly, in [27] an intelligent guide of a museum is modelled on the user interests and context behaviours exploiting Hierarchical Markov Models.

Nevertheless, first-order knowledge representation formalism is crucial to take into account more structured and complex features and relations involved in environment descriptions. A logical formalism for mining temporal patterns in a task of user modelling has been proposed in [17] in which the user behaviour is described according to the temporal sequences of his actions. The approach proposed in this paper allows us to tackle many complex scenarios such as context modelling, in which a situation and the actors involved in it evolve both in time and space. Some researchers proposed to model the user context with a n-dimensional space [26] and in particular they consider that 12 dimensions are sufficient to model the more relevant aspects in a context. Some of the dimensions they consider are: Absolute Time: a particular time interval in which events occur, Type Of Time: a non-absolute type of time period, such as "just after eating", Absolute Place: a particular location where events occur, such as "Paris", Type Of Place: a non-absolute type of place, such as "in bed". Starting from this type of information about the context, we should think to profile, for example, a user accessing to a room (home, office, museum, etc.) by describing contextual information (such as position in the room described by two spatial dimensions) and temporal information with the aim for the context to provide dynamics and adaptive functionalities in environments according to user habits.

2. Mining Multi-Dimensional Patterns

We used Datalog [42] as representation language for the domain knowledge and patterns, that here is briefly reviewed. For a more comprehensive introduction to logic programming and inductive logic programming we refer the reader to [10, 32, 24].

Definition 2.1. (Alphabet)
A first-order *alphabet* consists of a set of *constants*, a set of *variables*, a set of *function symbols*, and a non-empty set of *predicate symbols*. Each function symbol and each predicate symbol has a natural number (its *arity*) assigned to it.

The arity assigned to a function symbol represents the number of arguments the function has. Even if constants are defined as a separate class of symbols, it is convenient to view constants as function symbols of arity 0, and hence as a subclass of the set of function symbols.

Definition 2.2. (Terms)
A *term* is a constant symbol, a variable symbols, or an n-ary function symbol f applied to n terms $t_1, t_2, \dots, t_n$.

An atom $p(t_1, \dots, t_n)$ (or atomic formula) is a predicate symbol p of arity n applied to n terms t_i. Both l and its negation $\bar{l}$ are said to be *literals* (resp. positive and negative literal) whenever l is an atomic formula.

Definition 2.3. (Clause)
A *clause* is a formula of the form $\forall X_1 \forall X_2 \dots \forall X_n (L_1 \vee L_2 \vee \dots \vee \overline{L}_i \vee \overline{L}_{i+1} \vee \dots \vee \overline{L}_m)$ where each L_i is a literal and $X_1, X_2, \dots X_n$ are all the variables occurring in $L_1 \vee L_2 \vee \dots \overline{L}_i \vee \dots \overline{L}_m$. Most commonly the same clause is written as $L_1, L_2, \dots \leftarrow L_i, L_{i+1}, \dots L_m$.

Clauses, literals and terms are said to be *ground* whenever they do not contain variables. A *Horn clause* is a clause which contains at most one positive literal. A *Datalog clause* is a clause with no function symbols of non-zero arity; only variables and constants can be used as predicate arguments.

Definition 2.4. (Subsumption)
A *substitution* θ is defined as a set of bindings $\{X_1 \leftarrow a_1, \dots, X_n \leftarrow a_n\}$ where $X_i, 1 \leq i \leq n$ is a variable and $a_i, 1 \leq i \leq n$ is a term. A substitution θ is applicable to an expression e, obtaining the expression $e\theta$, by replacing all variables X_i with their corresponding terms a_i.

2.1. Multi-dimensional Relational Sequences

Now we introduce the definitions of multi-dimensional relational sequences.

Definition 2.5. (1-dimensional relational sequence)
A *1-dimensional relational sequence* may be defined as an ordered list of Datalog atoms separated by the operator $<$: $l_1 < l_2 < \dots < l_n$.

Example 2.1. The following list of ground Datalog atoms
`p(a,b)` $<$ `p(b,c)` $<$ `p(c,a)` $<$ `p(b,b)`
represents a 1-dimensional relational sequence. In general, for this kind of sequences, referring to one dimension only, the operator $<$ may be omitted as follows
`p(a,b) p(b,c) p(c,a) p(b,b)`
where it is implicit, for instance, that the atom `p(b,c)` follows the atom `p(a,b)`.

However, in order to make the proposed framework more general, the concept of *fluents* introduced by J. McCarthy in [30] should be considered: "After having defined a *situation*, s_t, as the complete state of the universe at an instant of time t, then a fluent is defined as a function whose domain is the space of situations. In particular, a *propositional fluent* ranges in (true,false). For example, raining(x, s_t) is true if and only if it is raining at the place x in the situation s_t."

If we consider a sequence as an ordered succession of events for each dimension, a fluent may be used to indicate that an atom is true for a given event. In particular, in our description language we can distinguish two kinds of Datalog atoms: *dimensional* and *non-dimensional* atoms. Specifically:

- non-dimensional atoms, that may be divided into

  - *fluent atoms*: explicitly referring to a given event (i.e., in which one of its argument denotes an event);
  - *non-fluent atoms*: denoting relations between objects (with arity greater than 1), or characterizing an object (with arity 1) involved in the sequence;

- dimensional atoms: referring to dimensional relations between events involved in the sequence.

Example 2.2. The following set of Datalog atoms

$\texttt{p(}e_1\texttt{,a,b) (}e_1 < e_2\texttt{) p(}e_2\texttt{,b,c) q(b,c)}$

denotes a 1-dimensional relational sequence with three non-dimensional atoms ($\texttt{p(}e_1\texttt{,a,b) p(}e_2\texttt{,b,c) q(b,c)}$) and one dimensional atom ($e_1 < e_2$). Specifically, $\texttt{p(}e_1\texttt{,a,b)}$ denotes the fluent `p(a,b)` at the event e_1, $\texttt{p(}e_2\texttt{,b,c)}$ denotes the fluent `p(b,c)` at the event e_2, ($e_1 < e_2$) indicates that the event e_2 is the direct successor of e_1 and `q(b,c)` represents a generic relation between the objects `b` and `c`.

Another way to read the previous example is the following: " `p(a,b)` is true in the event e_1, the event e_1 gives rise to the event e_2 where `p(b,c)` is true, and there is a relation `q` between `b` and `c`".

The choice to add the event as an argument of the predicates is necessary for the general case of n-dimensional sequences with $n > 1$. In this case, indeed, the operator $<$ is not sufficient to express multi-dimensional relations and we must use its general version $<_i, 1 \leq i \leq n$. Specifically, ($e_1 <_i e_2$) denotes that the event e_1 gives rise to the event e_2 in the dimension i. Hence, in our framework a multi-dimensional data is supposed to be a set of events, and to each dimension corresponds a sequence of events.

Definition 2.6. (Multi-dimensional relational sequence)
A *multi-dimensional relational sequence* is a set of Datalog atoms, involving k events and concerning n dimensions, in which there are non-dimensional atoms (fluents and non-fluents) and each event may be related to another event by means of the $<_i$ operators, $1 \leq i \leq n$.

After having defined what is a logical multi-dimensional sequence, in the following we give a detailed description of the dimensional operators used to describe multi-dimensional patterns.

2.1.1. Running Example: Cellular automaton data

To better explain the concepts, we will use as a running example the best-known example of a cellular automaton, named *The Game of Life*, devised by J.H. Conway in 1970 [12]. This simulation game resembles the processes of a society of living organisms.

The universe of the game involves a plan, assumed to be infinite, divided into cells, each of which is in one of two possible states, *live* – meaning that there is an organism – or *dead* . The idea is to start with a simple configuration of organisms and then observe how it changes as one applies the "genetic laws" for births, deaths, and survivals. Note that each cell of the plane has eight neighboring cells, four adjacent orthogonally and four adjacent diagonally. The rules are:

- **Births**: each empty cell adjacent to exactly three neighbours is a birth cell. An organism is placed on it in the next population;

- **Survivals**: every organism with two or three neighboring organisms survives for the next generation;

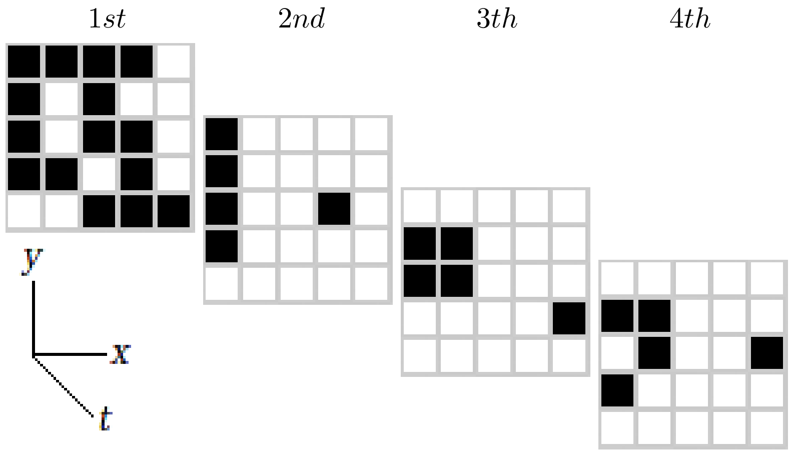

Figure 1. A sequence of evolving populations

- **Deaths**: each organism with four or more neighbours dies (is removed) for overpopulation. Every organism with one neighbour or none dies for isolation.

Note that all births and deaths occur simultaneously.

This example may be viewed as a case of a 3-dimensional sequence that combines spatial and temporal data. Indeed, we have a) the evolving process of each organism along the time (the first dimension), b) the adjacency's relation between organisms in horizontal sense and c) in vertical sense. Here the events are represented by the organisms that may be dimensionally related in three ways.

One can model the plane by using two dimensions (say x and y), while the time may be modeled by another dimension (say t). The plane containing the organisms has been viewed as a two-dimensional array. However, since in principle the plane is infinite, its left and right edges are considered to be stitched together, like the top and bottom edges, thus yielding a toroidal array. In particular, at time t an organism contained in the cell (i, j) of the (t)-th plane may be considered as an event that is *spatially* related with the cells $(i+1, j)$, $(i, j+1)$, $(i-1, j)$, $(i, j-1)$ of the (t)-th plane, and *temporally* related to the cells (i, j) of the $(t+1)$-th plane and (i, j) of the $(t-1)$-th plane.

In Figure 1 is reported a sequence of evolving populations, starting from an initial population of 15 organisms, that evolves along the time, based on the previously defined rules, in next populations. The following predicates may be used to describe this sequence: given a generic plane `live(`X`)` (resp. `dead(`X`)`) indicates that the organism X is live (resp. is dead); $A <_x B$ (resp. $A <_y B$) indicates that B is the next organism with respect to A in horizontal sense (resp. in vertical sense); and, $A <_t B$ indicates that A is the organism in the cell (i, j) of the generic (t)-th plane and B is the same organism in the cell (i, j) of the next $(t+1)$-th plane. In particular, the operators $<_x$ and $<_y$ indicate that an event is a direct successor, respectively, in horizontal and in vertical direction. While, the operator $<_t$ represents the direct successor of an event along the time dimension. The sequence reported in Figure 1 can be described, using this domain language, as follows:

```
/* 1st population */
  live(f1) (f1 <x f2) live(f2) ...
```

  ($f_1 <_y f_6$) live(f_6) ($f_6 <_x f_7$) ($f_2 <_y f_7$) dead(f_7) ...
  ($f_6 <_y f_{11}$) live(f_{11}) ($f_{11} <_x f_{12}$) ($f_7 <_y f_{12}$) dead(f_{12}) ...
/* 2nd population */
  live(s_1) ($s_1 <_x s_2$) dead(s_2) ...
  ($s_1 <_y s_6$) live(s_6) ($s_6 <_x s_7$) ($s_2 <_y s_7$) dead(s_7) ...
  ($s_6 <_y s_{11}$) live(s_{11}) ($s_{11} <_x s_{12}$) ($s_7 <_y s_{12}$) dead(s_{12}) ...
/* 3th population */
  ...
/* 4th population */
  ...
/* temporal relations */
  ($f_1 <_t s_1$) ($f_2 <_t s_2$) ($f_3 <_t s_3$) ($f_4 <_t s_4$) ($f_5 <_t s_5$)...

2.2. Multi-Dimensional Relational Patterns

In order to represent multi-dimensional relational patterns, some dimensional operators must be introduced. The following symbols for describing general event relationships along many dimensions have been adopted. In particular, given a set $\mathcal{D}$ of dimensions, in the following are reported the multi-dimensional operators:

- $<_i$: *next step on dimension* $i, \forall i \in \mathcal{D}$. This operator indicates the direct successor on the dimension i. For instance, $(x <_{time} y)$ denotes that the event y is the direct successor of the event x on the dimension *time*. `next_i/2` is the corresponding Datalog predicate used to denote the successor operator;

- $\lhd_i$: *after some steps on dimension* $i, \forall i \in \mathcal{D}$. This operator encodes the transitive closure of $<_i$. For example, $(y \lhd_{spatialx} z)$ states that the event z occurs somewhere after the event y on the dimension *spatialx*. `follow_i/2` is the corresponding Datalog representation;

- $\bigcirc_i^n$: *exactly after* n *steps on dimension* $i, \forall i \in \mathcal{D}$. In particular it calculates the n-th direct successor. For instance, $(x \bigcirc_{spatialz}^n w)$ states that the event w is the n-th direct successor of the event x on the dimension *spatialz*. The `followat_i/3` Datalog predicate is used to represent such a situation.

Note that, the dimensional characteristics in the sequences will be described by using the $<_i$ operator, while the two dimensional operators $\lhd_i$ and $\bigcirc_i^n$, will be used, in combination with the $<_i$ operator, to represent the frequent discovered patterns.

Example 2.3. With the above defined dimensional operators, an example of a simple temporal sequence could be:
  p(e_1,a,b) ($e_1 <_{time} e_2$) q(e_2,b,c) ($e_2 <_{time} e_3$) p(e_3,e,f) ($e_3 <_{time} e_4$) q(e_4,f,g)
and the relative temporal patterns that may be true when applied to it are
  p(E_1,X,Y) ($E_1 <_{time} E_2$) q(E_2,Y,Z)
  p(E_1,X,Y) ($E_1 \lhd_{time} E_2$) q(E_2,Z,W)
  p(E_1,X,Y) ($E_1 \bigcirc_{time}^2 E_2$) p($E_2$,Z,W)

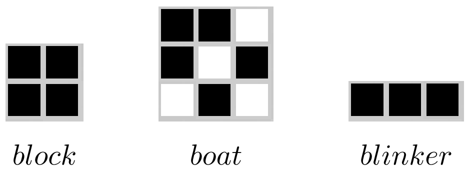

Figure 2. Some patterns occurring in the Game of Life

Definition 2.7. (Subsequence [18])
Given a sequence $\sigma = (e_1 e_2 \cdots e_m)$ of m elements, a sequence $\sigma' = (e'_1 e'_2 \cdots e'_k)$ of length k is a *subsequence* (or pattern) of the sequence σ if

1. $1 \leq k \leq m$
2. $\forall i, 1 \leq i \leq k, \exists j, 1 \leq j \leq m : e'_i = e_j$
3. $\forall i, j, 1 \leq i < j \leq k, \exists h, l, 1 \leq h < l \leq m : e'_i = e_h$ and $e'_j = e_l$.

The frequency of a subsequence in a sequence is the number of different mappings from elements of σ' into the elements of σ such that the previous conditions hold.

Note that this is a general definition of subsequence, in our case the *gaps* represented by the third condition are modelled by the $\lhd_i$ and $\bigcirc_i^n$ operators as reported in the following definition.

Definition 2.8. (Multi-dimensional relational pattern)
A *multi-dimensional relational pattern* is a set of Datalog atoms, involving k events and regarding n dimensions, in which there are non-dimensional atoms and each event may be related to another event by means of the operators $<_i$, $\lhd_i$ and $\bigcirc_i^n$ operators, $1 \leq i \leq n$.

Coming back to the cellular automaton data example, multi-dimensional patterns that may be discovered from some artificial populations include *still lifes* and *oscillators*, as those reported in Figure 2. In cellular automata, a still life is a pattern that does not change from one generation to the next, while, an oscillator is a pattern that returns to its original state, in the same orientation and position, after a finite number of generations. The *block* and *boat* patterns, reported in Figure 2, are still lifes, while the *blinker* is a two-phase oscillator (i.e., it returns to its original state after 2 generations). As reported in the following clausal descriptions of the patterns depicted in Figure 2, it is not necessary to express the temporal relations $<_t$ between all the possible organisms.

block still life
live(A) ($A <_x B$) live(B) ($A <_y C$) live(C) ($C <_x D$) ($B <_y D$) live(D) ($A <_t A'$) live(A') ($A' <_x B'$) live(B') ($A' <_y C'$) live(C') ($C' <_x D'$) ($B' <_y D'$) live(D')

boat still life
live(A) ($A <_x B$) live(B) ($A <_y C$) live(C) ($C \bigcirc_x^2 D$) live(D) ($B \bigcirc_y^2 E$) live(E) ($A <_t A'$) live(A') ($A' <_x B'$) live(B') ($A' <_y C'$) live(C') ($C' \bigcirc_x^2 D'$) live(D') ($B' \bigcirc_y^2 E'$) live(E')

blinker oscillator

live(A) ($A <_x B$) live(B) ($B <_x C$) live(C)
dead(D) ($D <_y B$) dead(E) ($B <_y E$) ($B <_t B'$) dead(A') ($A' <_x B'$) live(B')
($B' <_x C'$) dead(C') live(D') ($D' <_y B'$) live(E') ($B' <_y E'$)

We are interested in finding maximal frequent patterns with a high frequency in long sequences.

Definition 2.9. (Maximal pattern)
A pattern σ' of a sequence σ is *maximal* if there is no pattern σ'' of σ more frequent than σ' and such that σ' is a subsequence of σ''.

If we indicate the operator $<_i$ with the Datalog predicate `next_i(X,Y)`, the Datalog definition of the operators $\bigcirc_i^k$ and $\lhd_i$ can be formulated as follows:

```
followat_i(1,X,Y) ←
   next_i(X,Y), !.
followat_i(K,X,Y) ←
   next_i(X,Z),
   K1 is K - 1,
   followat_i(K1,Z,Y).

follow_i(X,Y) ←
   next_i(X,Y).
follow_i(X,Y) ←
   next_i(X,Z),
   follow_i(Z,Y).
```

These definitions are added to the background knowledge $\mathcal{B}$ and used to prove the dimensional operators appearing in the patterns using the following definition of subsumption.

Given S a multi-dimensional relational sequence, in the following we will indicate by Σ the set of Datalog clauses $\mathcal{B} \cup U$, where U is the set of ground atoms in S.

In particular given a sequence $S = (s_1, s_2, \ldots, s_n)$ the set Σ is made up of the following clauses:

```
/* definition of dimensional predicates */
followat_i(1,X,Y) ← ...
follow_i(X,Y) ← ...
/* atoms of the sequence */
s1 ←
s2 ←
...
sn ←
```

In order to calculate the frequency of a pattern over a sequence it is important to define the concept of sequence subsumption.

Definition 2.10. (Pattern Subsumption)
Given P a multi-dimensional relational pattern and S a multi-dimensional relational sequence, let $\Sigma =$

$\mathcal{B} \cup U$. The pattern P *subsumes* the sequence S, written as $P \subseteq S$, iff there exists an SLD$_{\mathrm{OI}}$-deduction of P from Σ.

An SLD$_{\mathrm{OI}}$-deduction is an SLD-deduction under Object Identity. In the Object Identity framework, within a clause, terms that are denoted with different symbols must be distinct, i.e. they must represent different objects of the domain.

2.3. The algorithm

After having defined the formalism for representing sequences and patterns, here we describe the algorithm for frequent multi-dimensional relational pattern mining based on the same idea of the generic level-wise search method, known in data mining from the APRIORI algorithm [1]. The level-wise algorithm makes a breadth-first search in the lattice of patterns ordered by a specialization relation $\preceq$. The search starts from the most general patterns, and at each level of the lattice the algorithm generates candidates by using the lattice structure and then evaluates the frequencies of the candidates. In the generation phase, some patterns are taken out using the monotonicity of pattern frequency (if a pattern is not frequent then none of its specializations is frequent).

The mining method is outlined in Algorithm 1. The generation of the frequent patterns is based on a top-down approach. The algorithm starts with the most general patterns. These initial patterns are all of length 1 and are generated by adding to the empty pattern a non-dimensional atom. Successively, at each step it tries to specialize all the potential frequent patterns, discarding the non-frequent patterns and storing the ones whose length is equal to the user specified input parameter *maxsize*. Furthermore, for each new refined pattern, semantically equivalent patterns are detected, by using the θ_{OI}-subsumption relation, and discarded. Note that the length of a pattern is defined as the number of non-dimensional atoms. In the specialization phase, the specialization operator under θ_{OI}-subsumption is used. Basically, the operator adds atoms to the pattern.

2.4. The background knowledge

The algorithm uses a background knowledge $\mathcal{B}$ (a set of Datalog clauses) containing the sequence and a set of constraints that must be satisfied by the generated patterns. In particular $\mathcal{B}$ contains:

- *maxsize(M)*: maximal pattern length (i.e., the maximum number of non-dimensional predicates that may appear in the pattern);
- *minfreq(m)*: this constraint indicates that the frequency of the patterns must be larger than m;
- *dimension(next_i)*: this kind of atom indicates that the sequence contains events on the dimension i. One can have more that one of such atoms, each of which denoting a different dimension. In particular, the number of these atoms represents the number of the dimensions.
- *type(p)*: denotes the type of the predicate's arguments p;
- *mode(p)*: denotes the input output mode of the predicate's arguments p;
- *negconstraint([$p_1, p_2, \ldots, p_n$])*: specifies a constraint that the patterns must not fulfill, i.e. if the clause $(p_1, p_2, \ldots, p_n)$ subsumes the pattern then it must be discarded. For instance, negconstraint([p(X,Y),q(Y)]) discards all the patterns subsumed by the clause (p(X,Y),q(Y));

Algorithm 1 MDLS

```
Require: $\Sigma = \mathcal{B} \cup U$, where $\mathcal{B}$ is the background knowledge and $U$ is the set of ground atoms in the
    sequence $S$.
Ensure: $P_{max}$: the set of maximal frequent patterns
 1: $P \leftarrow \{$ initial patterns $\}$
 2: $P_{max} \leftarrow \emptyset$
 3: while $P \neq \emptyset$ do
 4:   $P_s \leftarrow \emptyset$
 5:   for all $p \in P$ do
 6:     /* generation step */
 7:     $P_s \leftarrow P_s \cup$ {all the specializations of $p$ that satisfy all the constraints posconstraints, negcon-
        straints or atmostone}
 8:   $P \leftarrow \emptyset$
 9:   for all $p \in P_s$ do
10:     /* evaluation step */
11:     if freq($p$) $\geq$ minfreq then
12:       if length($p$) = maxsize then
13:         $P_{max} \leftarrow P_{max} \cup \{p\}$
14:       else
15:         $P \leftarrow P \cup \{p\}$
```

- *posconstraint([$p_1, p_2, \ldots, p_n$])*: specifies a constraint that the patterns must fulfill. It discards all the patterns that are not subsumed by the clause $(p_1, p_2, \ldots, p_n)$;
- *atmostone([$p_1, p_2, \ldots, p_n$])*: this constraint discards all the patterns that make true more than one predicate among $p_1, p_2, \ldots, p_n$. For instance, atmostone([red(X),blue(X),green(X)]) indicates that each constant in the pattern can assume at most one of red, blue or green value;
- *key([$p_1, p_2, \ldots, p_n$])*: it is optional and specifies that each pattern must have one of the non-dimensional predicates $p_1, p_2, \ldots p_n$ as a starting literal.

The use of the `dimension(next_i)` literals, that specify the number of dimensions the sequence is based on, allows to the corresponding definitions of the predicates `followat_i/3` and `follow_i/2` to be automatically generated and added to the background knowledge $\mathcal{B}$.

Classical mode and type declarations are used to specify a language bias indicating which predicates can be used in the patterns and to formulate constraints on the binding of variables. The solution space is further pruned by using some positive and negative constraints specified by the `negconstraint` and `posconstraint` literals. The last pruning choice is defined by the `atmostone` literals. This last constraint is able to describe that some predicates are of the same type.

Since each pattern a) must start with a non-dimensional predicate, or with a predefined key, and b) its frequency must be less than the sequence length, the frequency of a pattern can be defined as follows.

Definition 2.11. (Pattern Frequency)
Given a multi-dimensional relational pattern $P = (p_1, p_2, \ldots, p_n)$ and S a multi-dimensional relational sequence, the *frequency* of pattern P is equal to the number of different ground literals used in all the

possible SLD$_{\mathrm{OI}}$-deductions of P from $\Sigma = \mathcal{B} \cup U$ that make true the literal p_1.

Example 2.4. Given the following sequence $S \equiv p(e_1, a), q(a, t), q(a, s), (e_1 < e_2), p(e_2, b), q(b, a)$ and the pattern $P \equiv p(E, X), q(X, Y)$ there are 3 SLD$_{\mathrm{OI}}$-deductions of P from Σ with the OI substitutions $\theta_1 = \{E/e_1, X/a, Y/t\}$, $\theta_2 = \{E/e_1, X/a, Y/s\}$ and $\theta_3 = \{E/e_2, X/b, Y/a\}$. However, since θ_1 and θ_2 map the same constants to the variables of $p(E, X)$, the frequency of P on S is equal to 2.

2.5. The refinement step

The refinement of patterns is obtained by using a refinement operator ρ that maps each pattern to a set of specializations of the pattern, i.e. $\rho(p) \subset \{p' | p \preceq p'\}$ where $p \preceq p'$ means that p is more general of p' or that p subsumes p'. In particular, given the set $\mathcal{D}$ of dimensions, the set $\mathcal{F}$ of fluent atoms, the set $\mathcal{P}$ of non-fluent atoms, the refinement operator for specialazing the patterns is defined as follows:

adding a non-dimensional atom

- the pattern S is specialized by adding a non-dimensional atom $F \in \mathcal{F}$ (a fluent) referring to an event already introduced in S;
- the pattern S is specialized by adding a non-dimensional atom $P \in \mathcal{P}$;

adding a dimensional atom

- the pattern S is specialized by adding the dimensional atom $(x <_i y)$ $i \in \mathcal{D}$, relating the events x and y, iff $\exists$ a fluent $F \in \mathcal{F}$ in S which event argument is x and there not exist the atoms $(x \lhd_i y)$ and $(x \bigcirc_i^n y)$ in S;
- the pattern S is specialized by adding the dimensional atom $(x \lhd_i y)$ $i \in \mathcal{D}$, relating the events x and y, iff $\exists$ a fluent $F \in \mathcal{F}$ in S which event argument is x and there not exist the atoms $(x <_i y)$ and $(x \bigcirc_i^n y)$ in S;
- the pattern S is specialized by adding the dimensional atom $(x \bigcirc_i^n y)$ $i \in \mathcal{D}$, relating the events x and y, iff $\exists$ a fluent $F \in \mathcal{F}$ in S which event argument is x and there not exist the atoms $(x <_i y)$ and $(x \lhd_i y)$ in S.

The dimensional atoms are added iff there exists a fluent atom referring to its starting event. This is to avoid unuseful chains of dimensional predicates like this `p(`e_1`,a) (`$e_1 <_i e_2$`) (`$e_2 <_i e_3$`) (`$e_3 <_i e_4$`)`, that is naturally subsumed by `p(`e_1`,a) (`$e_1 \bigcirc_i^3 e_4$`)`.

We recall that the length of a pattern P is equal to the number of non-dimensional atoms in P.

3. Experiments

MDSL has been implemented in Yap Prolog and evaluated by making some experiments on an artificial dataset and on trace files collected from different users of Unix csh [14, 18].

Table 1. Warmr and MDLS performances (time in secs.).

		p_1		p_2		p_3		p_4		p_5	
		Warmr	MDLS	Warmr	MDLS	Warmr	MDLS	Warmr	MDLS	Warmr	MDLS
	1D	5,17	1,46	5,32	2,33	5,0	2,66	5,85	3,92	6,44	5,52
$\mathcal{P}1$	2D	3,75	1,37	4,23	1,97	4,22	2,84	3,68	3,38	4,36	4,59
	3D	3,98	1,32	3,46	1,80	4,00	2,72	4,08	3,47	4,02	4,04
	1D	5,82	1,53	12,22	3,14	26,52	5,83	45,84	9,3	63,79	13,59
$\mathcal{P}2$	2D	4,46	1,43	8,61	2,77	19,62	5,07	37,41	8,52	62,23	14,47
	3D	3,78	1,16	10,30	2,84	18,68	5,13	38,52	9,06	66,67	14,57

3.1. Artificial relational data

In order to generate synthetic data, a random problem generator has been implemented and used to generate multi-dimensional relational sequences. In particular, it randomly generates a sequence containing a frequent pattern taking as input the following parameters. The domain language is defined by a set $\mathcal{D}$ of d dimensions, a set $\mathcal{R}$ of r binary predicates, and a set $\mathcal{F}$ of f fluent predicates with arity 3. By using these predicates, a sequence, made up of Es events and Os objects, is generated by randomly selecting Rs relational literals and Fs fluent literals per event. A relational literal is generated by randomly selecting its predicate from $\mathcal{R}$ and randomly selecting its arguments from the set of Os objects. For each event, Fs fluent literals are generated by randomly selecting their predicates from $\mathcal{F}$ and randomly selecting its two relational arguments from the set of Os objects. The sequence contains $freq$ patterns with the same logical structure, made up of Ep events and Op objects. Each pattern contains Fp fluents literals per event and Rp relational literals randomly generated by using the above method.

Two problems, $\mathcal{P}_1$ and $\mathcal{P}_2$, have been generated, with r and f set to 3, Fs and Fp set to 1. In the former we fixed the length of the pattern, while in the latter we fixed the length of the sequence. In particular, the problem $\mathcal{P}_1$ has been divided into 5 sub-problems, where the number of events Es of the sequence has been set, respectively, to 100, 200, 300, 400 and 500, while the number of events Ep of the pattern has been fixed to 4. The problem $\mathcal{P}_2$ has been divided into 5 sub-problems, where the number of events Ep of the pattern has been set, respectively, to 4, 5, 6, 7 and 8, fixing the number of events Es of the sequence to 100. For each sub-problem 10 sequences have been generated.

Our system has been compared to Warmr [11], using the package ACE-ilProlog [8] kindly made available by Hendrik Blockeel. Table 1 reports the mean time, over the 10 sequences for each sub-problem (p_i, $1 \leq i \leq 5$), by executing both Warmr and MDLS. For each sub-problem of $\mathcal{P}_1$ we fixed $Ep = 4$, $Op = 3$ and $Rp = 2$, while the others parameter have been set, respectively, as follows $Es = 100, 200, 300, 400, 500$, $Os = 10, 20, 30, 40, 50$, $Rs = 40, 60, 80, 100, 120$, $freq = 10, 20, 30, 40, 50$. While, for the problem $\mathcal{P}_2$ we fixed $Es = 100$, $Os = 10$ and $Rs = 40$, $Ep = 4, 5, 6, 7, 8$, $Op = 3$, $Rp = 2$, $freq = 10$.

The first column of Table 1 indicates the kind of sequence (1D, 2D, 3D) for each problem, while the others the mean time in seconds for each corresponding sub-problem. As one can see, MDLS shows significant runtime improvements with respect to Warmr that is limited with respect to the length of the pattern. Indeed, the time increases as the length of the pattern grows, as reported for the problem $\mathcal{P}_2$.

3.2. Unix csh commands

The analysis of the use of Unix command shell represents one of the classic applications in the domain of adaptive user interfaces and user modelling. Greenberg [14] collected logs from 168 users of the unix csh, divided into 4 target groups: 55 novice programmers, 36 experienced programmers, 52 computer scientists and 25 non-programmers. Table 2 reports statistics of finding frequent patterns for 3 users logs from the Greenberg dataset.

Each Greenberg's log file corresponding to a user is divided into login sessions denoted by a starting and an ending time record. Each command entered in each session has been annotated with the current working directory, alias substitution, history use and error status. Furthermore, each command name may be followed by some options and some parameters. For instance the command `ls -a *.c` has name `ls`, option `-a` and parameter `*.c`.

As pointed out in [18], this problem is a relational problem, since commands are interrelated by their execution order (or time), and each command can be eventually related to one or more parameters. A shell log may be viewed as a 2-dimensional sequence, since each command is followed by another command (the first dimension) and each command line is composed by an ordered sequence of tokens (i.e., command name, options and parameters). Each shell log has been represented as a set of logical ground atoms as follows.

`command(e)` is the predicate used to indicate that `e` is a command. The command name has been used as a predicate symbol applied to `e`;

`parameter(e,p)` has been used to indicate that `p` is the parameter of `e`. The parameter name has been used as a predicate symbol applied to `p`;

`current_directory(c,d)` indicates that `d` is the current directory of the command `c`;

`next_c(c1,c2)` ($<_c$) indicates that the command `c2` is the direct command successor of `c1`;

`next_p(p1,p2)` ($<_p$) indicates that the parameter `p2` is the direct parameter successor of `p1`.

For instance the following shell log

```
cp paper.tex newpaper.tex
latex newpaper
xdvi newpaper
```

should be translated as

```
command(c1), '$cp'(c1),
   next_p(c1,c1p1), parameter(c1p1,'paper.tex'),
   next_p(c1p1,c1p2), parameter(c1p2,'newpaper.tex'),
next_c(c1,c2), '$latex'(c2),
   next_p(c2,c2p1), parameter(c2p1,'newpaper'),
next_c(c2,c3), '$xdvi'(c3),
   next_p(c3,c3p1), parameter(c3p1,'newpaper')
```

In this way it is possible to describe patterns such as

```
command(L), '$latex'(L),
   next_p(L,LP), parameter(LP,P),
next_c(L,X), '$xdvi'(X),
   next_p(X,XP), parameter(XP,P)
```

Table 2 reports statistics on MDLS performance on the Greenberg dataset. The first four columns denote, respectively, the user name, the number of literals of the sequence, the number of sessions for each user log file and the total number of commands in the log file. For each user some experiments has been made. The kind of experiment carried out is denoted in the fifth column that reports the operators used in the experiment. For instance $<_C$ and $<_P$ indicate that only these two operators have been used in the experiment. We see that, as the number of commands and dimensional operators grows, the execution time increases. Note that each session represents a sequence and a log file is a collection of sequences. There is no correlation between two sessions in a log file.

Table 2. MDLS performances (time in secs.). |S|: n. of literals in the sequence; |Ses|: n. of sessions for user log file; |C|: total number of commands in the log file; Op: dimensional operators used; L: max length of the patterns; F: min freq of the patterns; |MP|: n. of found maximal patterns; Sp: required specializations.

User	|S|	|Ses|	|C|	Op	L	F	Time	|MP|	Sp
				$<_C$	5	5	1.17	45	3597
				$<_C <_P$	5	5	2.68	45	9513
n9	2654	73	357	$<_C \bigcirc^n_C$	5	5	2.58	91	8495
				$<_C \lhd_C \bigcirc^n_C$	5	5	28.94	149	29081
				$<_{C,P} \lhd_{C,P} \bigcirc^n_{C,P}$	5	5	99.06	218	76299
				$<_C$	5	10	2.36	38	4512
				$<_C <_P$	5	10	3.58	18	6434
n17	5366	61	848	$<_C \bigcirc^n_C$	5	10	7.95	47	10808
				$<_C \lhd_C \bigcirc^n_C$	5	10	37.06	39	19198
				$<_{C,P} \lhd_{C,P} \bigcirc^n_{C,P}$	5	10	144.38	25	25142
				$<_C$	5	15	6.10	64	7716
				$<_C <_P$	5	15	19.33	77	24046
				$<_C \bigcirc^n_C$	5	15	16.30	139	20744
				$<_C \lhd_C \bigcirc^n_C$	5	15	138.06	162	51363
n7	12355	80	1231	$<_C$	5	80	1.78	9	953
				$<_C <_P$	5	80	4.06	11	2493
				$<_C \bigcirc^n_C$	5	80	4.14	16	2479
				$<_C \lhd_C \bigcirc^n_C$	5	80	42.55	29	6745
				$<_{C,P} \lhd_{C,P} \bigcirc^n_{C,P}$	5	80	164.99	49	17505

4. Related Work

As already pointed out, the problem of sequential pattern mining is a central one in a lot of data mining applications and many efforts have been done in order to propose purposely designed methods to face it.

Most of the works have been restricted to propositional patterns, that is, patterns not involving first order predicates. One of the early domains that highlights the need to describe with structural information the sequences was the bioinformatics. Thus, the need to represent many real world domains with structured data sequences became more unceasing, and consequently many efforts have been done to extend existing or propose new methods to manage sequential patterns in which first order predicates are involved. On the other hand, for a fair description of some application domains, the sequences must involve not only relational objects but also the evolution of each object in more than one dimension. Unfortunately, to our knowledge, there are no methods able to manage sequences whose description involves both relations and more than one dimension. In the following a brief survey of the techniques proposed to deal with relational or multi-dimensional sequences is presented along with the modelled application domain.

In [18] is presented a work, in the domain of user modelling, that helps shell users by creating scripts (a sequence of commands) from shell logs, that automate frequent performed tasks. The authors see this task as a relational learning problem, indeed commands may be interrelated by their execution order, and each command is possibly related to one or more parameters, giving out a representation of a shell log as a set of logical ground atoms. After having transformed shell logs in a relational representation, they applied the Warmr [11] system, an upgrade of the propositional Apriori algorithm that can detect first order logic association rules, for generating scripts. They used a specific predicate to specify that two commands are considered next to each other in a sequence.

Warmr [11] is based on the level-wise search of conventional association rule learning systems of the Apriori-family [1]. It extends these systems by looking for frequent patterns that may be expressed as conjunction of first-order literals. In Warmr a pattern is defined as a conjunction of first order literals. It performs a top-down level-wise search, starting with the key and refining patterns by adding literals to them. Infrequent patterns (i.e. patterns whose frequency is below a predefined threshold) are pruned as are their refinements. With Warmr it is possible to generate patterns that are syntactically different but semantically equivalent. This is due to the redundant conditions that may be added to a pattern or to the fact that the same pattern may be expressed in different ways.

As already described in [9], this problem may be avoided by using the Warmr's configurable language bias or by its constraint specification language. However, this solution does not solve the problem at all. Indeed, the constraints that may be defined in Warmr, by using its constraint specification language, are only syntax based, and they are not sufficient to handle semantic dependencies. For this and other limitations already described in [18, 17], in some cases Warmr system is not able to calculate frequent subsequences and it is difficult to correctly represent the specific sequence mining task.

In [25] are presented a logic language, SeqLog, for mining sequences of logical atoms, and the inductive mining system MineSeqLog, that combines principles of the level-wise search algorithm with the version space in order to find all patterns that satisfy a constraint by using an optimal refinement operator for SeqLog. SeqLog is a logic representational framework that adopts two operators to represent the sequences: one to indicate that an atom is the direct successor of another and the other to say that an atom occurs somewhere after another. Furthermore, based on this language, the notion of subsumption, entailment and a fix point semantic are given. However, with SeqLog one can represent unidimensional sequences only.

In this framework, however, the representation of the temporal dimension is represented by points in a straight line. Spirit-Log [29], an algorithm designed to mine first order temporal patterns, extends SeqLog by pushing regular expression constraints in the mining process. A further step aiming at facing with sequential patterns in which time is measured in terms of intervals instead of points is proposed in [5]. In this case, temporal pattern is defined as a set of atomic first order formulae where time is explicitly represented by an interval variable, together with a set of interval relationships (before, during, overlaps, start, finish, meet) described in terms of Allens' First Order Interval Logic [3]. Here, the temporal interval patterns aim at capturing how events taking place in time intervals and how they are related to each other. The process is performed by means of MILPRIT (Mining Interval Logic Patterns with Regular expressIons consTraints) algorithm which generalizes the idea of the SPIRIT algorithm introduced in [13] in the context of classical sequence patterns. In a high level, it follows the general Apriori strategy, working in steps, and each step producing patterns more specific than those produced in the previous step.

In [21] it is proposed an extension of classical Fisher kernels, working on sequences over flat alphabets, in order to make them able to model logical sequences, i.e., sequences over an alphabet of logical atoms. Fisher kernels were developed to combine generative models with kernel methods, and have shown promising results for the combinations of support vector machines with (logical) hidden Markov models and Bayesian networks. Successively, in [22] the same authors proposed an algorithm for selecting logical hidden Markov models from data. Hidden Markov models are one of the most popular methods for analyzing sequential data, but they can be exploited to handle sequence of flat/unstructured symbols. The proposed logical extension [23] overcomes such weakness by handling sequences of structured symbols by means of a probabilistic ILP framework.

The work above reported extend/propose techniques to mine sequences involving relational objects. However, these methods, both logical and propositional, do not mention the possibility to manage patterns in which more than one dimension is taken into account. On the other hand, it is not wrong to affirm that for most of the applications of real world domain generally the pattern sequences deal with different events each of which should be associated with different dimensions.

To this concern, the first work on mining multidimensional pattern sequence on unordered data is [19] followed by the work reported in [35], to mine both ordered and unordered data. In both these works, however, the concept of multi-dimensions only concerns the presence of (different) multiple attributes in the (single) data (table), e.g. the addition of different attributes to a transaction formes a multi-dimensional sequential dataset. Indeed, in [19] the authors investigate the possibility to discover association rules involving relationships among multiple attributes, that they call dimensions, using a data cube structure [16]. At the same way, Pinto et al in [35] consider multi-dimensional sequential pattern mining as the process of mining one or more dimensions of information, i.e. attributes, in which the order of the dimension values is not important. Here two multi-dimensional sequential pattern mining algorithms HYBRID and PSFP are proposed. HYBRID uses the BUC [7] traversal system to find the next dimension value combination to process. Then the subsets of data records that contain this dimension value combination are found, and finally a sequential pattern miner algorithm PrefixSpan [33] is run to find the sequential patterns in this dataset. On the contrary, PSFP (PrefixSpan with Frequent Pattern growth) finds all the sequential patterns in the full dataset once, but grows, and subsequently mines an FP-tree alongside each pattern.

A more recent attempt to deal with multi-dimensional sequences is proposed in [45] and [43]. In the former, the problem of finding multi-dimensional sequential patterns is partitioned into two subproblems: mining multi-dimensional patterns and mining sequential patterns. The BUC-like algorithm [7] is used to mine multi-dimensional patterns and discover the frequent dimensional items that successively will be used to find the frequent dimensional two-itemsets pattern and so on until there is not further longer dimensional frequent patterns. The process is carried out for each dimensional item until all the frequent dimensional itemsets patterns are discovered. Successively, the original sequence database is projected to construct a new sequential database with those tuples that contain those frequent dimensional itemsets. The sequential pattern miner algorithm PrefixSpan [33] is used to mine sequential patterns on the projected database. In [43], two algorithms, AprioriMD and PrefixMDSpan, extending respectively the Apriori and PrefixSpan algorithms to the case of multi-dimensional data, are presented. In this work the sequences are grouped according to the dimension they belong to. Successively, for each dimension, the sequence mining algorithms are executed and finally the resulting frequent patterns for each dimension are "merged" into a unique multi-dimensional pattern.

However, all the works in multi-dimensional data mining have been restricted to the propositional case, not involving a first-order representation formalism and considering the concept of multi-dimensions as the presence of more attributes in the sequential pattern.

5. Conclusions

The issue of discovering sequential patterns from sequence data have drawn a lot of research efforts both in single data table and in multiple data table, known as multi-relational data. Although much work has been done in the area, no previous research revealed ways to find sequential patterns from multidimensional sequence data and in particular sequential patterns from multi-dimensional sequence expressed in first-order logic. Indeed, some works faces the problem of knowledge discovery from spatial and temporal data in the multi-relational data mining research area but there exists no contributions to manage the general case of multi-dimensional data in which, for example, spatial and temporal information may co-exist. Other works on multi-dimensional data mining, in some cases thinking to the concept of multi-dimension of a sequence as the presence of multiple attributes in data descriptions, have been restricted to the propositional case, not involving a first-order representation formalism. Finally, other works propose a (two-dimensional) knowledge representation formalism to represent spatio-temporal information based on multi-dimensional modal logics.

In this paper we proposed a logical framework for mining multi-dimensional patterns in which many dimensions can be specified. What we can obtain are maximal frequent multi-dimensional patterns described in a first order language. One of the most important characteristic of using logical framework for sequences is that we can incorporate additional information by using a background knowledge, and that any relation between atoms can be expressed or learned. The result is a dedicated system in which are incorporated specific language bias for multi-dimensional data in order to rise a faster execution and a smaller search space.

Acknowledgements

The authors would like to thank Jan Ramon for giving useful suggestions on setting the system Warmr and Saul Greenberg for providing the test Unix log data.

References

[1] Agrawal, R., Manilla, H., Srikant, R., Toivonen, H., Verkamo, A.: Fast discovery of association rules, in: *Advances in Knowledge Discovery and Data Mining* (U. Fayyad, G. Piatetsky-Shapiro, P. Smyth, R. Uthurusamy, Eds.), AAAI Press, 1996, 307–328.

[2] Agrawal, R., Srikant, R.: Mining sequential patterns, in: *Proceedings of the Int. Conf. on Data Engineering (ICDE95)*, 1995, 3–14.

[3] Allen, J., Ferguson, G.: *Actions and Events in Interval Temporal Logic*, Technical Report TR521, University of Rochester Rochester, NY, USA, 1994.

[4] de Amo, S., Furtado, D.: First-order temporal pattern mining with regular expression constraints, *Data & Knowledge Engineering*, **62**(3), 2007, 401–420.

[5] de Amo, S., Giacometti, A., Junior, W. P.: Mining First-Order Temporal Interval Patterns with Regular Expression Constraints, in: *Proceedings of the 9th International Conference on Data Warehousing and Knowledge Discovery* (I. Y. Song, J. Eder, T. M. Nguyen, Eds.), vol. 4654 of *LNCS*, Springer, 2007, 459–469.

[6] Bennett, B., Cohn, A. G., Wolter, F., Zakharyaschev, M.: Multi-Dimensional Modal Logic as a Framework for Spatio-Temporal Reasoning, *Applied Intelligence*, **17**(3), 2002, 239–251.

[7] Beyer, K. S., Ramakrishnan, R.: Bottom-Up Computation of Sparse and Iceberg CUBEs, in: *Proceedings ACM SIGMOD International Conference on Management of Data* (A. Delis, C. Faloutsos, S. Ghandeharizadeh, Eds.), ACM Press, 1999, 359–370.

[8] Blockeel, H., Dehaspe, L., Demoen, B., Janssens, G., Ramon, J., Vandecasteele, H.: Executing query packs in ILP, in: *Proceedings of the 10th International Conference on Inductive Logic Programming* (J. Cussens, A. Frisch, Eds.), vol. 1866 of *LNAI*, Springer, 2000, 60–77.

[9] Blockeel, H., Fürnkranz, J., Prskawetz, A., Billari, F.: Detecting temporal changes in event sequences: An application to demographic data, in: *Proceedings of the 5th European Conference on Principles of Data Mining and Knowledge Discovery* (L. D. Raedt, A. Siebes, Eds.), vol. 2168, Springer, 2001, 29–41.

[10] Bratko, I.: *Prolog programming for artificial intelligence, 3rd ed.*, Addison-Wesley Longman Publishing Co., 2001, ISBN 0-201-40375-7.

[11] Dehaspe, L., Toivonen, H.: Discovery of frequent Datalog patterns, *Data Mining and Knowledge Discovery*, **3**(1), 1999, 7–36.

[12] Gardner, M.: The fantastic combinations of John Conway's new solitaire game "life", *Scientific American*, **2**(223), October 1970, 120–123.

[13] Garofalakis, M. N., Rastogi, R., Shim, K.: SPIRIT: Sequential Pattern Mining with Regular Expression Constraints, in: *Proceedings of 25th International Conference on Very Large Data Bases* (M. P. Atkinson, M. E. Orlowska, P. Valduriez, S. B. Zdonik, M. L. Brodie, Eds.), Morgan Kaufmann, 1999, 223–234.

[14] Greenberg, S.: Using Unix: collected traces of 168 users, Research Report 88/333/45, Department of Computer Science, University of Calgary, Alberta, 1988.

[15] Han, J., Pei, J., Yin, Y.: Mining frequent patterns without candidate generation, in: *Proceedings of the 2000 ACM SIGMOD international conference on Management of data*, ACM, 2000, 1–12.

[16] Harinarayan, V., Rajaraman, A., Ullman, J. D.: Implementing Data Cubes Efficiently, in: *Proceedings of the 1996 ACM SIGMOD International Conference on Management of Data* (H. V. Jagadish, I. S. Mumick, Eds.), ACM Press, 1996, 205–216.

[17] Jacobs, N.: *Relational Sequence Learning and User Modelling*, Ph.D. Thesis, Department of Computer Science, K.U.Leuven, Leuven, Belgium, October 2004.

[18] Jacobs, N., Blockeel, H.: From shell logs to shell scripts, in: *Proceedings of the 11th International Conference on Inductive Logic Programming* (C. Rouveirol, M. Sebag, Eds.), vol. 2157, Springer, 2001, 80–90.

[19] Kamber, M., Han, J., Chiang, J.: Metarule-Guided Mining of Multi-Dimensional Association Rules Using Data Cubes, in: *Proceeding of the 3rd International Conference on Knowledge Discovery and Data Mining*, 1997, 207–210.

[20] Kaplan, C. A., Fenwick, J., Chen, J.: Adaptive Hypertext Navigation Based On User Goals and Context, *User Modeling and User-Adapted Interaction*, **3**(3), 1993, 193–220.

[21] Kersting, K., Gärtner, T.: Fisher Kernels for Logical Sequences, in: *Proceedings of the 15th European Conference on Machine Learning* (J.-F. Boulicaut, F. Esposito, F. Giannotti, D. Pedreschi, Eds.), vol. 3201 of *LNCS*, Springer, 2004, 205–216.

[22] Kersting, K., Raedt, L. D., , Raiko, T.: Logical Hidden Markov Models, *Journal of Artificial Intelligence Research*, **25**, 2006, 425–456.

[23] Kersting, K., Raiko, T.: 'Say EM' for Selecting Probabilistic Models for Logical Sequences, in: *Proceedings of the 21st Conference on Uncertainty in Artificial Intelligence* (F. Bacchus, T. Jaakkola, Eds.), AUAI Press, 2005, 300–307.

[24] Lavrac, N., Dzeroski, S.: *Inductive Logic Programming: Techniques and Applications.*, Ellis Horwood, New York, 1994.

[25] Lee, S., De Raedt, L.: Constraint based mining of first order sequences in SeqLog, in: *Database Support for Data Mining Applications* (R. Meo, P. Lanzi, M. Klemettinen, Eds.), vol. 2682 of *LNCS*, Springer, 2004, 155–176.

[26] Lenat, D.: *The Dimensions of Context-Space*, Cycorp, 1998.

[27] Liao, L., Patterson, D. J., Fox, D., Kautz, H. A.: Learning and inferring transportation routines, *Artificial Intelligence*, **171**(5-6), 2007, 311–331.

[28] Malerba, D., Lisi, F.: Discovering associations between spatial objects: An ilp application, in: *Proceedings of the 11th International Conference on Inductive Logic Programming*, vol. 2157 of *LNCS*, Springer, 2001, 156–166.

[29] Masson, C., Jacquenet, F.: Mining frequent logical sequences with SPIRIT-LoG, in: *Proceedings of the 12th International Conference on Inductive Logic Programming* (S. Matwin, C. Sammut, Eds.), vol. 2583 of *LNAI*, Sringer, 2003, 166–181.

[30] McCarthy, J., Hayes, P.: Some Philosophical Problems from the Standpoint of Artificial Intelligence, in: *Machine Intelligence 4* (B. Meltzer, D. Michie, Eds.), Edinburgh University Press, 1969, 463–502.

[31] Moyle, S., Muggleton, S.: Learning Programs in the Event Calculus, in: *Proceedings of the 7th International Workshop on Inductive Logic Programming*, Springer, 1997, 205–212.

[32] Muggleton, S., De Raedt, L.: Inductive Logic Programming: Theory and Methods, *Journal of Logic Programming*, **19/20**, 1994, 629–679.

[33] Pei, J., Han, J., Mortazavi-Asl, B., Pinto, H., Chen, Q., Dayal, U., Hsu, M.-C.: PrefixSpan: Mining Sequential Patterns Efficiently by Prefix-Projected Pattern Growth, *Proceedings of the 17th International Conference on Data Engineering*, 2001, 215–226.

[34] Pei, P., Han, J., Wang, W.: Mining Sequential Patterns with Constraints in Large Databases, in: *Proceedings of the 11th ACM International Conference on Information and Knowledge Management*, 2002, 18–25.

[35] Pinto, H., Han, J., Pei, J., Wang, K., Chen, Q., Dayal, U.: Multi-dimensional sequential pattern mining, in: *Proceedings of the tenth international conference on Information and knowledge management*, ACM Press, 2001, 81–88.

[36] Popelínsky, L.: Knowledge Discovery in Spatial Data by Means of ILP, in: *Proceedings of the Second European Symposium on Principles of Data Mining and Knowledge Discovery*, Springer, 1998, 185–193.

[37] Randell, C., Muller, H.: Context Awareness by Analysing Accelerometer Data, in: *The Fourth International Symposium on Wearable Computers* (B. MacIntyre, B. Iannucci, Eds.), IEEE Computer Society, 2000, ISBN 1530-0811, 175–176.

[38] Rodríguez, J., Alonso, C., Böstrom, H.: Learning first order logic time series classifiers, in: *Proceedings of the 10th International Workshop on Inductive Logic Programming* (J. Cussens, A. Frisch, Eds.), Springer, 2000, 260–275.

[39] Si, H., Kawahara, Y., Morikawa, H., Aoyama, T.: A Stochastic Approach for Creating Context-Aware Services based on Context Histories in Smart Home, in: *Proceeding of the 3rd International Conference on Pervasive Computing, Exploiting Context Histories in Smart Environments*, 2005, 37–41.

[40] Siewiorek, D., Smailagic, A., Furukawa, J., Krause, A., Moraveji, N., Reiger, K., Shaffer, J., Wong, F. L.: SenSay: A Context-Aware Mobile Phone, in: *Proceedings of the 7th IEEE International Symposium on Wearable Computers*, IEEE, 2003, 248–249.

[41] Souchon, N., Limbourg, Q., Vanderdonckt, J.: Task Modelling in Multiple Contexts of Use, in: *Proceedings of the 9th International Workshop on Interactive Systems. Design, Specification, and Verification*, Springer, 2002, ISBN 3-540-00266-9, 59–73.

[42] Ullman, J.: *Principles of Database and Knowledge-Base Systems*, vol. I, Computer Science Press, 1988.

[43] Yu, C.-C., Chen, Y.-L.: Mining Sequential Patterns from Multidimensional Sequence Data, *IEEE Transactions on Knowledge and Data Engineering*, **17**(1), 2005, 136–140, ISSN 1041-4347.

[44] Zaki, M.: SPADE: An efficient algorithm for mining frequent sequences, *Machine Learning Journal: Special issue on Unsupervised Learning*, **42**(1/2), 2001, 31–60.

[45] Zhao, Q., Bhowmick, S.: *Sequential pattern mining: a survey*, Technical report, Center for Advanced Information Systems, School of Computer Engineering, Nanyang Technological University, Singapore, 2003.