Mining Frequent Patterns from Multi-Dimensional Relational Sequences

Nicola Di Mauro, Teresa M.A. Basile, Stefano Ferilli, and Floriana Esposito

Università degli Studi di Bari, Dipartimento di Informatica, 70125 Bari, Italy

Abstract. The problem addressed in this paper regards the discovering of frequent multi-dimensional patterns from relational sequences. 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. 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. A complete framework and an *Inductive Logic Programming* algorithm to tackle this problem is presented with preliminary experiments focussing on artificial multi-dimensional sequences.

1 Introduction

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 as showed by the research efforts produced in recent years [1, 22, 7, 17, 18]. Sequential information may concern data on multiple dimensions and, hence, the mining of sequential patterns from multi-dimensional information results very important. The first work on mining multi-dimensional patterns has been presented in 2001 by Pinto et al. [18]. However, all the works in multi-dimensional data mining have been restricted to the propositional case, not involving a first-order representation formalism.

Some works facing the problem of knowledge discovery from spatial and temporal data in the multi-relational data mining research area [15, 19, 5, 20, 12] are present in literature, 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.

In this paper an Inductive Logic Programming (ILP) [16] algorithm for discovering *first-order* (DATALOG) *maximal frequent patterns* in *multi-dimensional* relational sequences is provided. 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 (next, follow, follow-at).

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 [9, 11, 13, 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 logical representation gives us the possibility to encode temporal, spatial and other dimensional spaces without requiring to discriminate between them. Furthermore, it is possible to represent any other domain relations and let them to co-exist with other dimensional ones.

An interesting application of multi-dimensional logical pattern mining is modelling. A logical formalism for mining temporal patterns in a task of user modelling has been proposed in [8] 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. For instance, we should think to profile 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.

2 Mining Multi-Dimensional Patterns

We used DATALOG [21] as representation language for the domain knowledge and patterns, that here is briefly reviewed. A *term* is defined as a constant symbol or a variable. An atom $p(t_1, \ldots, t_n)$ is a predicate p of arity n applied to n terms t_i. A *substitution* θ is defined as a set of bindings $\{X_1 \leftarrow a_1, \ldots, X_n \leftarrow a_n\}$ 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.

Definition 1. *A 1-dimensional relational sequence may be defined as an ordered list of* DATALOG *atoms separated by the operator* $<$, $l_1 < l_2 < \cdots < l_n$.

Example 1. The following list of DATALOG atoms

p(a,b) < p(b,c) < p(c,a) < p(b,b)

represents a 1-dimensional sequence. In general, for this kind of sequences, referring to one dimension only, the operator $<$ can be omitted as follows

p(a,b) p(b,c) p(c,a) p(b,b)

where is implicit, for instance, that the atom p(b,c) follows the atom p(a,b).

Hovewer, in order to make the proposed framework more general, the concept of *fluents* introduced by J. McCarthy in [14] 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."

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. The following set of DATALOG atoms

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

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

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

The choice to add the event as an argument of the predicates is necessary in 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. *A multi-dimensional relational sequence is a set of* DATALOG *atoms, involving k events and regarding 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 Multi-Dimensional Patterns

In order to represent multi-dimensional patterns, some dimensional operators must be introduced. The following symbols for describing general event relationships along many dimensions has 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*. $\mathsf{next\_i}/2$ 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*. follows_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 follows_at_1/3 Datalog predicate is used to represent such a situation.

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 $<_i$ operator, to represent the frequent discovered patterns.

Example 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)

In general, 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 σ if for a given $h < m - k$: $e_{h+i} = e'_{i+1}$, $1 \leq i \leq k$. The frequency of a subsequence in a sequence is the number of all the possible values of h such that the previous condition holds.

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

Definition 3. *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,* $1 \leq i \leq n$.

2.2 The algorithm

In this section we describe the algorithm for frequent pattern discovery 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 main 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-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-subsumption is used. Basically, the operator adds atoms to the pattern.

Algorithm 1 MDLS

Require:
maxsize: maximal pattern length (i.e., the maximum number of non-dimensional predicates appearing in the pattern);
minfreq: the threshold;
Ensure: P_{max}: the set of maximal frequent patterns

```
1: P ← { initial patterns }
2: P_max ← ∅
3: while P ≠ ∅ do
4:    P_s ← ∅
5:    for all p ∈ P do
6:       {generation step}
7:       P_s ← P_s ∪ {all the specializations of p }
8:    P ← ∅
9:    for all p ∈ P_s do
10:      {evaluation step}
11:      if freq(p) ≥ minfreq then
12:         if length(p) = maxsize then
13:            P_max ← P_max ∪ {p}
14:         else
15:            P ← P ∪ {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 specializing 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 with x as

its event argument 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 with x as its event argument 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 with x as its event argument 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 $\mathsf{p}(e_1,\mathsf{a})$ $(e_1 <_i e_2)$ $(e_2 <_i e_3)$ $(e_3 <_i e_4)$, that is naturally subsumed by $\mathsf{p}(e_1,\mathsf{a})$ $(e_1 \bigcirc_i^3 e_4)$.

As regards the language bias, classical mode and type declarations are used to specify which predicates can be used in patterns and to formulate constraints on the binding of variables.

3 Experiments

In order to evaluate the proposed technique we made preliminary experiments, applying the algorithm to a simple example about 3D data and to an artificial dataset.

3.1 3D example: Cellular automaton data

In the following we evaluated the algorithm on the best-known example of a cellular automaton, named *The Game of Life*, devised by J.H. Conway in 1970 [6]. This simulation game resembles the processes of a society of living organisms.

The universe of the game involves a plane, 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 neighbors 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;
- **Deaths**: each organism with four or more neighbors dies (is removed) for overpopulation. Every organism with one neighbor or none dies for isolation.

Note that all births and deaths occur simultaneously.

We 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. Hovewer, since the plane is in principle infinite, its left and right edges are considered to be stitched together, like the top and bottom edges, thus yielding a toroidal array.

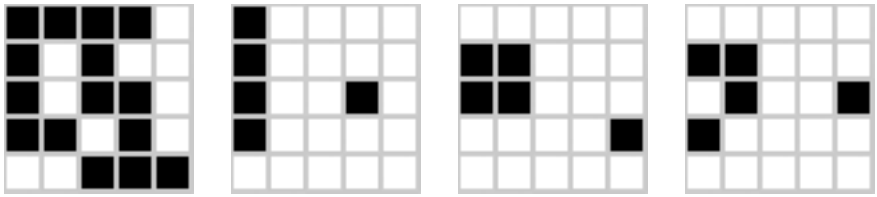

Fig. 1. A sequence of evolving populations

In Figure 1 is reported a sequence of evolving populations, form an initial population of 25 organisms, that can be described in the defined domain language as follows.

/* 1st population */
live(f_1) ($f_1 <_x f_2$) live(f_2) ...
($f_1 <_y f_6$) live(f_6) ($f_6 <_x f_7$) ($f_2 <_y f_7$) dead(f_7) ...
($f_6 <_y f_{11}$) live($f_1 1$) ($f_1 1 <_x f_{12}$) ($f_7 <_y f_{12}$) 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($s_1 1$) ($s_1 1 <_x s_{12}$) ($s_7 <_y s_{12}$) dead(s_{12}) ...
/* 3th population */
...
/* 4th population */
...
/* temporal relations */
($f_1 <_t s_1$) ($f_2 <_t s_2$) ($f_3 <_t s_3$) ($f_4 <_t s_4$) ($f_5 <_t s_5$)...

We used the operators $<_x$ and $<_y$ to indicate that an event is a direct successor, respectively, in horizontal and in vertical direction. While, the operator $<_t$ represents direct successor of an event along the time dimension. In particular, this kind of 3-dimensional sequences combines spatial and temporal data.

Executing the algorithm on some artificial population we obtained many different multi-dimensional patterns, including *still lifes*, and *oscillators*[1] as that reported in Figure 2. The *block* and *boat* patterns are still lifes, while the *blinker* is a two-phase oscillator.

[1] 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.

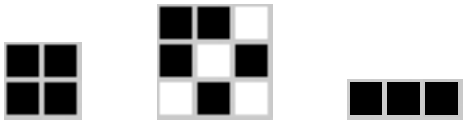

Fig. 2. Some patterns occur in the Game of Life

block

$\mathsf{live}(A)\ (A <_x B)\ \mathsf{live}(B)\ (A <_y C)\ \mathsf{live}(C)\ (C <_x D)\ (B <_y D)\ \mathsf{live}(D)$
$(A <_t A')\ \mathsf{live}(A')\ (A' <_x B')\ \mathsf{live}(B')\ (A' <_y C')\ \mathsf{live}(C')\ (C' <_x D')$
$(B' <_y D')\ \mathsf{live}(D')$

boat

$\mathsf{live}(A)\ (A <_x B)\ \mathsf{live}(B)\ (A <_y C)\ \mathsf{live}(C)\ (C \bigcirc_x^2 D)\ \mathsf{live}(D)\ (B \bigcirc_y^2 E)$
$\mathsf{live}(E)\ (A <_t A')\ \mathsf{live}(A')\ (A' <_x B')\ \mathsf{live}(B')\ (A' <_y C')\ \mathsf{live}(C')\ (C' \bigcirc_x^2 D')$
$\mathsf{live}(D')\ (B' \bigcirc_y^2 E')\ \mathsf{live}(E')$

blinker

$\mathsf{live}(A)\ (A <_x B)\ \mathsf{live}(B)\ (B <_x C)\ \mathsf{live}(C)$
$\mathsf{dead}(D)\ (D <_y B)\ \mathsf{dead}(E)\ (B <_y E)\ (B <_t B')\ \mathsf{dead}(A')\ (A' <_x B')$
$\mathsf{live}(B')\ (B' <_x C')\ \mathsf{dead}(C')\ \mathsf{live}(D')\ (D' <_y B')\ \mathsf{live}(E')\ (B' <_y E')$

3.2 Artificial relational data

In order to evaluate the proposed algorithm on more convincing 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 kind of 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

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	5,0t, F0	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

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 [5], using the package ACE-ilProlog [2] 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 $Rs = 40$, $Ep = 4, 5, 6, 7, 8$, $Op = 3$, $Rp = 2$, $freq = 10$. t, F 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 outperforms 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 the problem $\mathcal{P}_2$.

4 Related Work

In this section we will review some recent work on mining logical sequences.

In [9] 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 [5] 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 [5] 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 [3], 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 [9, 8], 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 [11] 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 [10] has been 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 [10] overcomes such weakness by handling sequences of structured symbols by means of a probabilistic ILP framework. However, the mining of multi-dimensional sequence is not taken into account from these methods both logical and propositional.

5 Conclusions

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. Preliminary experimental results prove the validity of the proposed approach.

Acknowledgements This work is partially founded by the Italian COFIN project "Learning Hierarchical, Abstract Models from Temporal/Spatial Data". The authors would like to thank Jan Ramon for giving useful suggestions on setting the system Warmr.

References

1. R. Agrawal, H. Manilla, R. Srikant, H. Toivonen, and A.I. Verkamo. Fast discovery of association rules. In U.M. Fayyad, G. Piatetsky-Shapiro, P. Smyth, and R. Uthurusamy, editors, *Advances in Knowledge Discovery and Data Mining*, pages 307–328. AAAI Press, 1996.
2. H. Blockeel, L. Dehaspe, B. Demoen, G. Janssens, J. Ramon, and H. Vandecasteele. Executing query packs in ilp. In J. Cussens and A. Frisch, editors, *Proceedings of the 10th International Conference on Inductive Logic Programming*, volume 1866 of *LNAI*, pages 60–77. Springer, 2000.
3. H. Blockeel, J. Fürnkranz, A. Prskawetz, and F. Billari. Detectin temporal change in event sequences: an application to demographic data. In *Proceedings of the 5th European Conference on Principles of Data Mining and Knowledge Discovery*, pages 29–41. Springer, 2001.
4. S. de Amo and D.A. Furtado. First-order temporal pattern mining with regular expression constraints. *Data & Knowledge Engineering*, 62(3):401–420, 2007.
5. L. Dehaspe and H. Toivonen. Discovery of frequent datalog patterns. *Data Mining and Knowledge Discovery*, 3(1):7–36, 1999.
6. M. Gardner. The fantastic combinations of John Conway's new solitaire game "life". *Scientific American*, 2(223):120–123, October 1970.
7. J. Han, J. Pei, and Y. Yin. Mining frequent patterns without candidate generation. In *ACM-SIGMOD Int. Conf.Management of Data (SIGMOD'00)*, pages 1–12, 2000.
8. N. Jacobs. *Relational Sequence Learning and User Modelling*. PhD thesis, Department of Computer Science, K.U.Leuven, Leuven, Belgium, October 2004.
9. N. Jacobs and H. Blockeel. From shell logs to shell scripts. In C. Rouveirol and M. Sebag, editors, *Proceedings of the 11th International Conference on Inductive Logic Programming*, volume 2157, pages 80–90. Springer, 2001.
10. K. Kersting and T. Raiko. 'Say EM' for selecting probabilistic models for logical sequences. In F. Bacchus and T. Jaakkola, editors, *Proceedings of the 21st Conference on Uncertainty in Artificial Intelligence (UAI05)*, pages 300–307. AUAI Press, 2005.
11. S.D. Lee and L. De Raedt. Constraint based mining of first order sequences in SeqLog. In R. Meo, P.L. Lanzi, and M. Klemettinen, editors, *Database Support for Data Mining Applications*, volume 2682 of *LNCS*, pages 155–176. Springer, 2004.
12. D. Malerba and F.A. Lisi. Discovering associations between spatial objects: An ilp application. In *Proceedings of the 11th International Conference on Inductive Logic Programming*, volume 2157 of *LNCS*, pages 156–166. Springer, 2001.
13. C. Masson and F. Jacquenet. Mining frequent logical sequences with SPIRIT-LoG. In *Proceedings of the 12th International Conference on Inductive Logic Programming*, volume 2583 of *LNAI*, pages 166–181. Sringer Verlag, 2003.
14. J. McCarthy and P.J. Hayes. Some philosophical problems from the standpoint of artificial intelligence. In B. Meltzer and D. Michie, editors, *Machine Intelligence 4*, pages 463–502. Edinburgh University Press, 1969. reprinted in McC90.

15. S. Moyle and S. Muggleton. Learning programs in the event calculus. In *Proceedings of the 7th International Workshop on Inductive Logic Programming*, pages 205–212. Springer, 1997.
16. S. Muggleton and L. De Raedt. Inductive logic programming: Theory and methods. *Journal of Logic Programming*, 19/20:629–679, 1994.
17. P.J. Pei, J. Han, and W. Wang. Mining sequential patterns with constraints in large databases. In *Proceedings of the 11th ACM International Conference on Information and Knowledge Management*, pages 18–25, 2002.
18. H. Pinto, J. Han, J. Pei, K. Wang, Q. Chen, and U. Dayal. Multi-dimensional sequential pattern mining. In *CIKM '01: Proceedings of the tenth international conference on Information and knowledge management*, pages 81–88, New York, NY, USA, 2001. ACM Press.
19. L. Popelínsky. Knowledge discovery in spatial data by means of ILP. In *Proceedings of the Second European Symposium on Principles of Data Mining and Knowledge Discovery*, pages 185–193. Springer, 1998.
20. J. Rodríguez, C. Alonso, and H. Böstrom. Learning first order logic time series classifiers. In J. Cussens and A. Frisch, editors, *Proceedings of the 10th International Workshop on Inductive Logic Programming*, pages 260–275. Springer, 2000.
21. J.D. Ullman. *Principles of Database and Knowledge-Base Systems*, volume I. Computer Science Press, 1988.
22. M.J. Zaki. SPADE: An efficient algorithm for mining frequent sequences. *Machine Learning Journal: Special issue on Unsupervised Learning*, 42(1/2):31–60, 2001.