

# Unsupervised Learning of Sequential Patterns

Andreas D. Lattner and Otthein Herzog  
TZI – Center for Computing Technologies  
Universität Bremen, PO Box 330 440,  
28334 Bremen, Germany  
[adl|herzog]@tzi.de

## Abstract

*In order to allow agents for acting autonomously and making their decisions on a solid basis an interpretation of the current scene has to be done. Scene interpretation can be done by checking if certain patterns match to the current belief of the world. In some cases it is impossible to acquire all situations an agent might have to deal with later at run-time in advance. Here, an automated acquisition of patterns would help an agent to adapt to the environment. Agents in dynamic environments have to deal with world representations that change over time. Additionally, predicates between arbitrary objects can exist in their belief of the world. In this work we present a learning approach that learns temporal patterns from a sequence of predicates.*

## 1. Introduction

In order to allow agents for acting autonomously and making their decisions on a solid basis, an interpretation of the current scene has to be done. Scene interpretation can be done by checking if certain patterns match to the current belief of the world. If intentions of other agents or events that are likely to happen in the future can be recognized the agent's performance can be improved as it can adapt the behavior to the situation. Example domains are intelligent vehicles or robots in the RoboCup leagues [10, 11].

In some cases it is impossible or too time-consuming to acquire all situations an agent might have to deal with at run-time in advance, or it is necessary that the agent adapts to a certain situation. It might even not be known what situations an agent will encounter in the future. In these cases an automated acquisition of patterns would help an agent to adapt to the environment.

We focus on qualitative representations as they allow for a concise representation of the important information. Such a representation allows to use background knowledge, to plan future actions, to recognize plans of other agents, and

is comprehensible for humans the same time. In our approach we map quantitative data to qualitative representations. Time series are divided into different segments which satisfy certain monotonicity or threshold conditions as it is suggested by Miene et al. [10, 11]. E.g., if the distance between two objects is observed it can be divided into increasing and decreasing distance representing approaching and departing relations (cf. [11]).

Agents in dynamic environments have to deal with world representations that change over time. Additionally, predicates between arbitrary objects can exist in their belief of the world. Current learning approaches do not handle these requirements properly. In this work we present a learning approach that learns temporal patterns from a sequence of predicates.

The paper is organized as follows: Section 2 presents work related to ours. The representation of scenes and patterns is described in Sections 3 and 4. Section 5 addresses the learning of patterns. A first evaluation of our approach is presented in Section 6. The paper closes with a conclusion and some ideas for future work.

## 2. Related Work

There are many approaches in the areas of association rule mining, inductive logic programming, and spatial, temporal, or spatiotemporal data mining which are relevant to the work presented here. The most important ones are presented in this section.

Association rule mining addresses the problem of discovering association rules in data. One famous example is the mining of rules in basket data [1]. Agrawal et al. and Zaki et al. present fast algorithms for mining association rules [2, 14]. These approaches have been developed for the mining of association rules in item sets. Thus, no temporal relationships between the items are represented.

Mannila et al. extended association rule mining by taking event sequences into account [8]. They describe al-

gorithms which find all relevant episodes which occur frequently in the event sequence.

Höppner presents an approach for learning rules about temporal relationships between labeled time intervals [5]. The labeled time intervals consist of propositions. Similar to our work interval relationships are described by Allen’s interval logic [3]. The major difference is that predicates which represent relations between certain objects are not considered in their work.

Other researchers in the area of spatial association rule mining allow for more complex representations with variables but do not take temporal interval relations into account (e.g., [6, 7, 9]).

The work presented here combines ideas from different directions. Similar to Höppner’s work [5] the learned patterns describe temporal interrelationships with Allen’s interval logic. Contrary to Höppner’s approach our representation allows for describing predicates between different objects similar to approaches like [7]. The generation of frequent patterns comprises a top-down approach starting from the most general pattern and specializing it. At each level of the pattern mining just the frequent patterns of the previous step are taken into account knowing that only combinations of frequent patterns can result in frequent patterns again. This is a typical approach in association rule mining (e.g., [8]).

Tan et al. present different interestingness measures for association patterns [13]. In our work some new evaluation criteria for the more complex representation are defined.

### 3. Scene Representation

A dynamic scene is represented by a set of objects and predicates between these objects. The predicates are only valid for certain time intervals and the scene can thus be considered as a sequence of (spatial or conceptual) predicates. These predicates are in specific temporal relations regarding the time dimension. Miene et al. presented an approach how to create such a representation [11].

Each predicate  $r$  is an instance of a predicate definition  $rd$ . We use the letter  $r$  for predicates/relations; the letter  $p$  is used for patterns.  $\mathcal{R}_{schema} = \{rd_1, rd_2, \dots\}$  is the set of all predicate definitions  $rd_i := \langle l_i, a_i \rangle$  with label  $l_i$  and arity  $a_i$ , i.e., each  $rd_i$  defines a predicate between  $a_i$  objects<sup>1</sup>. Predicates can be hierarchically structured. If a predicate definition  $rd_1$  specializes another predicate definition  $rd_2$  all instances of  $rd_1$  are also instances of the super predicate  $rd_2$ .

<sup>1</sup>It would also be possible to specify more formally what objects are including instance-of relations, attributes, etc.; this is omitted here in order to reduce complexity for the moment. So far we just see objects as uniquely identifiable entities.

If we have to handle more than one dynamic scene, let  $S = \{s_1, s_2, \dots\}$  be the set of the different sequences  $s_i$ . A single sequence  $s_i$  is defined as

$$s_i = (\mathcal{R}_i, \mathcal{TR}_i, \mathcal{C}_i)$$

where  $\mathcal{R}_i$  is the set of predicates,  $\mathcal{TR}_i$  is the set of temporal relations and  $\mathcal{C}_i$  is the set of constants representing different objects in the scene. Each predicate is defined as  $r(c_1, \dots, c_n)$  with  $r$  being an instance of  $rd_i \in \mathcal{R}_{schema}$ , having arity  $n = a_i$ , and  $c_{i,1}, \dots, c_{i,n} \in \mathcal{C}_i$  are representing the objects where the predicate holds. The set of temporal relations  $\mathcal{TR}_i = \{tr_1, tr_2, \dots\}$  defines relations between pairs of elements in  $\mathcal{R}_i$ . Each temporal relation is defined as  $tr_i(r_a, op, r_b)$  with  $r_a, r_b \in \mathcal{R}_i$  and  $op \in \{<, =, >, d, di, o, oi, m, mi, s, si, f, fi\}$ , i.e., Allen’s temporal relations between intervals [3].

Note that it can happen that the same set of objects appears in different instances of an identical type of predicate at different times. These predicates must be represented by different predicate instances as they usually do not have identical temporal interrelations with other predicates, e.g., two vehicles which are in the relation `faster(v1, v2)` in two different time intervals  $i_1$  and  $i_2$ . Therefore, we assign IDs  $r_i$  to the different predicates. An example is:

```
r1 = behind(obj1, obj2)
r2 = accelerates(obj1)
r3 = leftOf(obj2, obj3)
tr1 = r1 o r2
tr2 = r1 < r3
tr3 = r2 < r3
```

Here, three predicates exist between `obj1`, `obj2`, and `obj3` (`behind`, `accelerates`, and `leftOf`). These predicates have certain temporal restrictions: `r1` must be before `r3`, `r2` must be before `r3`, and `r1` overlaps `r2`.

### 4. Pattern Description

This section introduces the representation of patterns, how patterns are to be matched in sequences, and how patterns can be specialized or generalized.

#### 4.1. Pattern Representation

Patterns are abstract descriptions of sequence parts with specific properties. A pattern defines what predicates must occur and how their temporal interrelationship has to be. Let  $\mathcal{P} = \{p_1, p_2, \dots\}$  be the set of all patterns  $p_i$ . A pattern is (similar to sequences) defined as

$$p_i = (\mathcal{R}_i, \mathcal{TR}_i, \mathcal{V}_i).$$

$\mathcal{R}_i$  is the set of predicates  $r_{ij}(v_{ij,1}, \dots, v_{ij,n})$  with  $v_{ij,1}, \dots, v_{ij,n} \in \mathcal{V}_i$ .  $\mathcal{V}_i$  is the set of all variables used in

the pattern.  $\mathcal{TR}_i$  defines the set of the temporal relations which have already been defined above.

An example for such a pattern description is the following abstraction of the sequence above (capital letters stand for variables, lower case letters for constants):

```
r1 = accelerates(X)
r2 = behind(X, Y)
tr1 = r1 o r2
```

In this pattern there are two predicates and  $r1$  overlaps  $r2$ .

## 4.2. Pattern Matching

A pattern  $p$  matches in a (part of a) sequence  $sp$  if there exists a mapping of a subset of the constants in  $sp$  to all variables in  $p$  such that all predicates defined in the pattern exist between the mapped objects and all time constraints of  $p$  are satisfied by the time intervals in the sequence. In order to restrict the exploration region a window size can be defined. Only matches within a certain neighborhood (specified by the window size) are valid. In our case the window size determines the number of adjacent start and end time points of different predicates which are taken into account for matching a pattern.

During the pattern matching algorithm a sliding window is used, and at each position of the window all matches for the different patterns are collected. A match consists of the position in the sequence and an assignment of objects to the variables of the pattern.

## 4.3. Specialization and Generalization of Patterns

Different patterns can be put into generalization-specialization relations. A pattern  $p_1$  subsumes another pattern  $p_2$  if it covers all sequence parts which are covered by  $p_2$ :

$$p_1 \sqsubseteq p_2 := \forall sp, matches(p_2, sp) : matches(p_1, sp).$$

If  $p_1$  additionally covers at least one sequence part which is not covered by  $p_2$  it is more general:

$$p_1 \sqsubset p_2 := p_1 \sqsubseteq p_2 \wedge \exists sp_x : matches(p_1, sp_x), \neg matches(p_2, sp_x).$$

This is the case if  $p_1 \sqsubseteq p_2 \wedge p_1 \not\sqsubseteq p_2$ .

In order to specialize a pattern it is possible to add a new predicate  $r$  to  $\mathcal{R}_i$ , add a new temporal relation  $tr$  to  $\mathcal{TR}_i$ , unify two variables, or specialize a predicate, i.e., replacing it with another more special predicate. Accordingly it is possible to generalize a pattern by removing a predicate  $r$  from  $\mathcal{R}_i$ , removing a temporal relation  $tr$  from  $\mathcal{TR}_i$ , inserting a new variable, or generalizing a predicate  $r$ , i.e., replacing it with another more general predicate.

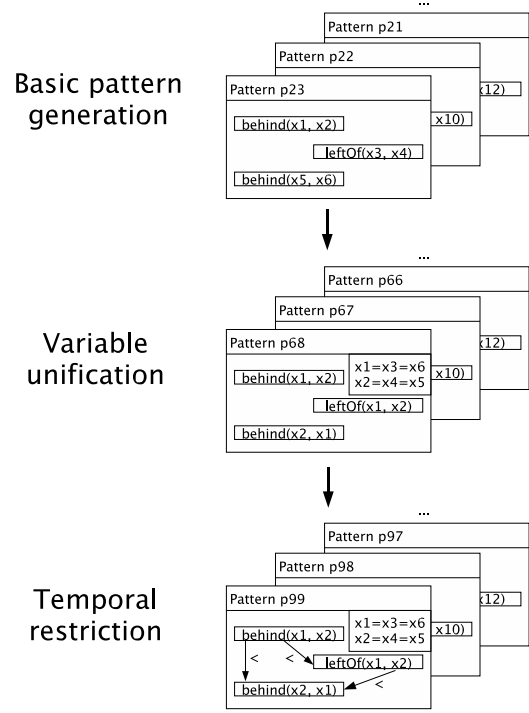


Figure 1. Pattern generation

At the specialization of a pattern by adding a predicate a new instance of any of the predicate definitions can be added to the pattern with variables which are not used in the pattern so far. If a pattern is specialized by adding a new temporal relation for any pair of predicates in the pattern (which has not been constrained so far) a new temporal restriction can be added. A specialization through variable unification can be done by unifying an arbitrary pair of variables. Another specialization would be to replace a predicate by a more special predicate. These different specialization steps can be used at the top-down induction of rules as described in Section 5.1.

## 5. Unsupervised Learning of Sequential Patterns

Unsupervised learning of patterns in a sequence  $S$  (or in a set of sequences  $S$ ) is more difficult than learning from examples as there is no clear task to separate positive from negative examples. The task is to find “interesting” patterns which appear repeatedly. Unless there is no more information how to identify the patterns of interest, it just can be searched for patterns that appear frequently in the sequence of data.

The following subsections present the learning algorithm and evaluation criteria how to estimate the quality of a pattern.

## 5.1. Top-down Induction

In order to create frequent patterns top-down induction is applied. In the first step all frequent patterns of the desired pattern size (i.e., number of used predicates) are created. Here, for all predicates new variables are created. In the second step, the patterns are specialized by the unification of variables, i.e., the different predicates can be “connected” via identical variables after this step. In the third step temporal restrictions are added to the patterns. If a maximum value for the number of patterns to keep (*maxPatterns*) is set, after each step only the best patterns are kept. The basic steps are illustrated in Figure 1. The pattern generation algorithm is:

### 1. Basic pattern generation.

- (a) Create initial patterns (consisting of just one predicate each) and store them in the *currentPatternsList*. Set *currentLevel* to 1.
- (b) Remove all patterns from the *currentPatternsList* which do not satisfy certain conditions (e.g., minimum frequency). Add all patterns in *currentPatternsList* to *allPatternsList* if *currentLevel* < *minPatternLevel*.
- (c) If *currentLevel* ≤ *maxPatternLevel* create next level patterns: Clear *currentPatternsList*, specialize all frequent patterns by combining them with all predicates of the frequent patterns of the previous step and store them in the *currentPatternsList*. Keep the best *maxPatterns* patterns in the *allPatternsList*. Go to step 1b.

### 2. Specialize by variable unification.

- (a) Set the *currentPatternList* to *allPatternList*.
- (b) For each pattern in *currentPatternList* create all possible variable unifications and store them in the cleared *currentPatternList*.
- (c) Remove all patterns from the *currentPatternsList* which do not satisfy certain conditions (e.g., minimum frequency). Add all patterns in *currentPatternsList* to *allPatternsList*.
- (d) If new patterns were created go to step 2b.
- (e) Keep the best *maxPatterns* patterns in the *allPatternsList*.

### 3. Specialize by temporal relations.

- (a) Set the *currentPatternList* to *allPatternList*.
- (b) For each pattern create statistics about temporal relations between predicates (i.e., count the occurrences of the different interval relations for each pattern pair).
- (c) If the creation of a temporal relation between two predicates in a pattern does not violate certain conditions (e.g., minimum frequency), the pattern is copied, specialized by this temporal relation, and added to the cleared *currentPatternList*.
- (d) Add all patterns in *currentPatternsList* to *allPatternsList*. If new patterns were created go to step 3b.
- (e) Keep the best *maxPatterns* patterns in the *allPatternsList*.

The parameters of the algorithm are:

- **minPatternLevel:** The minimum size of the patterns (i.e., the minimum number of predicates appearing in each pattern).
- **maxPatternLevel:** The maximum size of the patterns (i.e., the maximum number of predicates appearing in each pattern).
- **maxPatterns:** The maximum number of patterns to be kept after each step of the algorithm.

## 5.2. Pattern Evaluation

In order to evaluate the patterns (and to keep the *k* best patterns) an evaluation function was defined. Right now our evaluation function takes five different values into account: relative pattern size, relative pattern frequency, coherence, temporal restrictiveness, and predicate preference. Further criteria can be added if necessary.

The relative pattern size is the pattern size divided by the maximum pattern size:

$$relPatSize = \frac{patSize}{maxPatSize}$$

This measure prefers larger patterns. The assumption here is that patterns with more predicates which are still frequent contain more useful information than shorter ones.

The relative pattern frequency is the number of matches multiplied by the pattern size relative to the length of the whole sequence:

$$relPatFrq = \frac{patSize \cdot numMatches}{sequenceLength}$$

Patterns which appear more often are assigned a higher value.

The coherence gives information how “connected” the predicates in the pattern are by putting the number of used variables in relation to the maximum possible number of variables for a pattern:

$$coh = 1 - \frac{numVariables}{maxNumberVariables}$$

Here, the idea is that patterns which are highly connected through variables provide more information because they are interrelated by shared objects. Currently we just regard binary predicates, thus

$$maxNumberVariables = 2 \cdot patSize.$$

The temporal restrictiveness is the number of restricted predicate pairs in the pattern to the maximum number of time restrictions:

$$tmpRestr = \frac{numRestrictedPredPairs}{maxNumRestrictedPredPairs} = \frac{numRestrictedPredPairs}{\frac{patSize \cdot (patSize - 1)}{2}}$$

Again it is assumed to be advantageous if the pattern is more specialized. A higher number of temporal restrictions leads to a higher value.

Predicate preference computes the mean of all used predicate preferences:

$$prPref = \frac{1}{patSize} \sum_{i=1}^{patSize} preference(predicate_i)$$

This is needed if certain predicates should be privileged, e.g., if some predicates are assumed to be more useful than others.

The overall evaluation function of a pattern computes a weighted sum of the five values:

$$eval = \frac{a \cdot relPatSize + b \cdot relPatFrq + c \cdot coh + d \cdot tmpRestr + e \cdot prPref}{a + b + c + d + e}$$

All single measures and the overall pattern quality have values between 0 and 1.

## 6. Evaluation

This section presents a first evaluation of our learning algorithm. First, the idea is shown by applying the algorithm to a simple example. After that experiments on some more complex data with varying parameters are presented. Finally, some statements about the complexity of the current algorithm are given.

```
s; behind; a; b e; approaches; x; y
s; faster; x; y s; behind; c; a
e; behind; a; b e; behind; c; a
s; leftOf; a; b s; behind; a; d
e; leftOf; a; b s; follows; x; y
e; faster; x; y e; behind; a; d
s; behind; b; a s; leftOf; a; d
e; behind; b; a e; leftOf; a; d
s; behind; a; c e; follows; x; y
s; approaches; x; y s; behind; d; a
e; behind; a; c e; behind; d; a
s; leftOf; a; c s; faster; x; z
e; leftOf; a; c e; faster; x; z
```

Figure 2. Input for sequence example

Step	New patterns	Kept patterns	All patterns
Initial patterns	5	5	0
Pattern level 2	15	10	10
Pattern level 3	28	13	23
Unification 1	255	57	80
Unification 2	393	51	131
Unification 3	200	15	146
Unification 4	32	1	147
Unification 5	1	0	147
Temp. Rest. 1	13	13	160
Temp. Rest. 2	21	21	181
Temp. Rest. 3	9	9	190

Table 1. Number of patterns in the different steps of the algorithm

### 6.1. Simple Example

In this subsection patterns for a simple artificial example are created. In the example there are seven different objects (a - d and x - z), five predicates (behind, faster, leftOf, approaches, and follows). The input data is shown in Figure 2. The first column gives the information if the interval starts or ends in this row. The names of the predicates are in the second column. The third and fourth columns show the IDs of the objects for which the predicate holds. The temporal order is represented by the rows from top to bottom.

Figure 3 shows the different intervals in this example. As it can be seen, object a has behind and leftOf relations to the objects b, c, and d. The predicates with x, y, and z are not important; they are just “noise” in the example.

The program was started with these parameters:

- $minPatternSize = 2$ ,
- $maxPatternSize = 3$ , and
- $maxPatterns = \infty$  (i.e., keep all created patterns).

The weights of the evaluation function were set to:  $a = 0.2$ ,  $b = 0.4$ ,  $c = 0.2$ ,  $d = 0.2$ , and  $e = 0.0$ . The predicate preference has not been realized so far and is thus not taken into account here.

Table 1 shows the number of patterns for the different steps. Note that patterns are just kept if they have more than

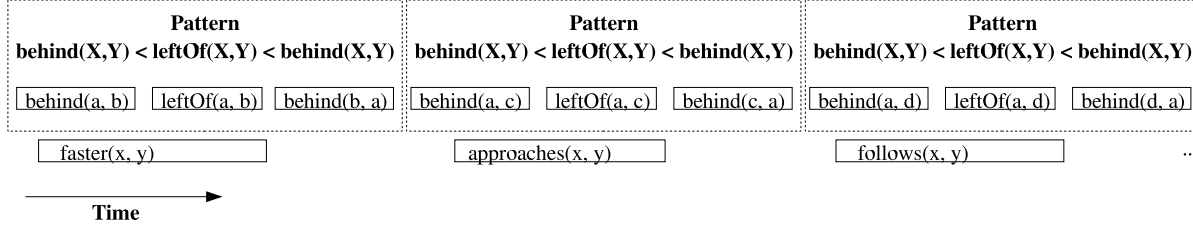


Figure 3. Example sequence and matched pattern

Pattern	Overall quality
behind(X <sub>9</sub> X <sub>29</sub> )[r <sub>1</sub> ], leftOf(X <sub>9</sub> X <sub>29</sub> )[r <sub>2</sub> ], behind(X <sub>29</sub> X <sub>9</sub> )[r <sub>3</sub> ], r <sub>1</sub> <->r <sub>2</sub> : before, r <sub>1</sub> <->r <sub>3</sub> : before, r <sub>2</sub> <->r <sub>3</sub> : before	0.780
behind(X <sub>9</sub> X <sub>29</sub> )[r <sub>1</sub> ], leftOf(X <sub>9</sub> X <sub>29</sub> )[r <sub>2</sub> ], behind(X <sub>29</sub> X <sub>9</sub> )[r <sub>3</sub> ], r <sub>1</sub> <->r <sub>2</sub> : before, r <sub>1</sub> <->r <sub>3</sub> : before	0.713
behind(X <sub>9</sub> X <sub>29</sub> )[r <sub>1</sub> ], leftOf(X <sub>9</sub> X <sub>29</sub> )[r <sub>2</sub> ], behind(X <sub>29</sub> X <sub>9</sub> )[r <sub>3</sub> ], r <sub>1</sub> <->r <sub>3</sub> : before, r <sub>2</sub> <->r <sub>3</sub> : before	0.713
behind(X <sub>9</sub> X <sub>29</sub> )[r <sub>1</sub> ], leftOf(X <sub>9</sub> X <sub>29</sub> )[r <sub>2</sub> ], behind(X <sub>29</sub> X <sub>9</sub> )[r <sub>3</sub> ], r <sub>1</sub> <->r <sub>2</sub> : before, r <sub>2</sub> <->r <sub>3</sub> : before	0.713
behind(X <sub>9</sub> X <sub>29</sub> )[r <sub>1</sub> ], leftOf(X <sub>9</sub> X <sub>29</sub> )[r <sub>2</sub> ], behind(X <sub>29</sub> X <sub>9</sub> )[r <sub>3</sub> ], r <sub>2</sub> <->r <sub>3</sub> : before	0.646

Table 2. Five best patterns

one predicate. The five best patterns are shown in Table 2. The intended pattern in the example was a simplified overtaking scenario (first, the overtaking car is behind another car, then it is left of, and finally the objects in the behind predicate are switched). As it can be seen, the intended pattern was found and evaluated best. Matched sequence parts are illustrated in Figure 3, enclosed by a dashed line.

## 6.2. Experiments

We evaluated the algorithm on an artificial data set with different settings. In order to create random data sets we wrote a program which generates input data for our learning program. Different parameters allow for creating different test sets. The parameters are: number of predicates, number of constants, fraction of “real patterns” in the random data, and the length of the sequence. As the “real pattern” the one above ( $\text{behind}(X, Y)$ ,  $\text{leftOf}(X, Y)$ ,  $\text{behind}(Y, X)$ ) was used with varying constants for  $X$  and  $Y$  and some random predicates inside the pattern.

For the experiments random sequences with 500 predicates (i.e., 1000 start and end time points) were created. The evaluation function was the same as shown above in Section 6.1. The window size was set to 15 (large enough to find the intended pattern). The number of constants (representing objects) was five in all settings. The minimum frequency in order to keep a pattern was 0.1. Figure 4 shows the number of created and kept patterns for the different settings in the experiments.

The number of predicates was varied in the first experi-

ment (5, 7, 9, and 11). One interesting fact is that with an increasing number of predicates the number of created patterns decreases. This seems counterintuitive at first glance. This phenomenon can be explained by the randomly created data. The more predicates are used the less frequent patterns with (random) predicates are created. This explains the decreasing number of created patterns in the first experiment.

In the second experiment the maximum size of the patterns was set to different values (2 - 5). This experiment shows the problem of complexity. With the increasing maximal size of patterns the number of created patterns grows exponentially. Future modifications of the algorithm should address this problem.

In the third experiment the number of patterns to keep after each step of the algorithm was varied ( $\text{maxPatterns} = 5, 10, 15, 20$ ). It shows how the number of created (and kept) patterns can be reduced by just keeping a certain number of patterns after each step of the basic pattern generation. The number of kept patterns almost stays constant. This might be due to the fact that many of the bad performing patterns are not created as their “parents” are pruned early while the important patterns are kept after each step.

## 6.3. Complexity

The algorithm consists of three main steps. The basic pattern generation algorithm creates predicate combinations up to an upper bound (the pattern size). If all predicate combinations were created the number of created patterns would be equal to the number of possibilities to take  $k$  elements out of  $n$  elements ignoring the order and allowing to draw elements repeatedly. Let  $n$  be the number of predicates and  $k$  the maximum pattern size. Then the number of basic patterns to be created is:

$$\#basicPatterns = \binom{n+k-1}{k}$$

Table 3 shows the number of possible patterns for different pattern sizes if five predicates exist, i.e.,  $n = 5$ .

The variable unification problem is similar to the problem of creating all possible clusters of arbitrary sizes for  $n$  objects. This leads to the Bell number [4]. The number

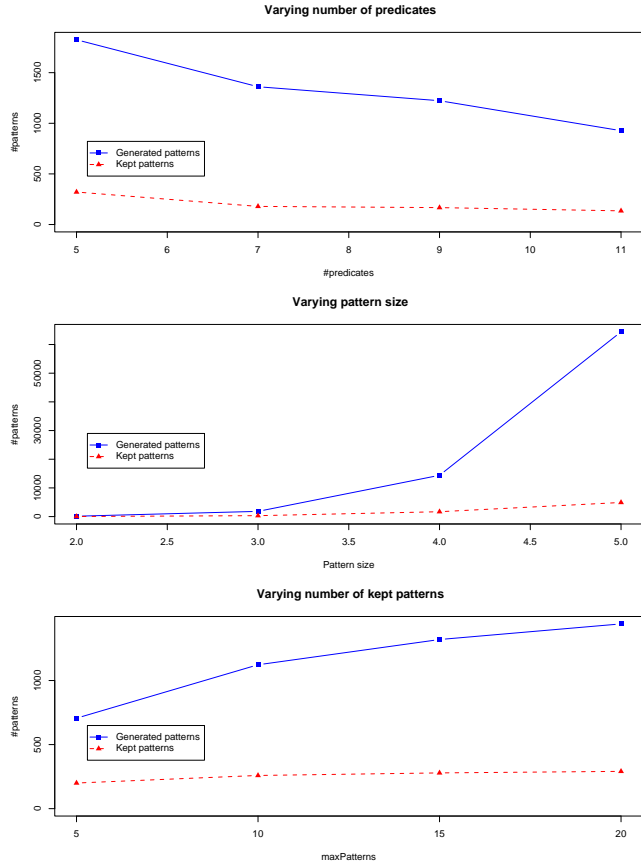


Figure 4. Evaluation results

Size of pattern	1	2	3	4	5
#possiblePatterns	5	15	35	70	126

Table 3. Number of possible patterns with five predicates

of all possible variable combinations for  $n$  variables can be computed by the formula (cf. [12]):

$$\begin{aligned} \#posVarUnific &= \sum_{k=1}^n \left\{ \begin{matrix} n \\ k \end{matrix} \right\} \\ &= \sum_{k=1}^n \frac{1}{k!} \sum_{j=1}^k (-1)^{k-j} \binom{k}{j} j^n \end{aligned}$$

The Bell numbers for different  $n$  are shown in Table 4.

The number of possible temporal restrictions for a pattern depends on the number of predicates used in the pattern

n	1	2	3	4	5	6	7	8	9	10
$B_n$	1	2	5	15	52	203	877	4140	21147	115975

Table 4. Bell numbers

Size of pattern	1	2	3	4	5
#possibleRestrictions	1	14	$14^3$	$14^6$	$14^{10}$

Table 5. Number of possible temporal restrictions with five predicates

and the number of temporal relations. It is possible to assign one of Allen’s 13 relations or no relation to each predicate pair. Of course, not all combinations are possible (e.g., if  $p_1$  is before  $p_2$  and  $p_2$  is before  $p_3$  there  $p_1$  must be before  $p_3$ ). For simplicity, we assume independency between the predicate pairs for the moment. In that case for each predicate pair 14 variations are possible. Thus, we have to calculate the power of 14 to the number of possible predicate pairs ( $n$  is the number of predicates in the pattern):

$$\begin{aligned} \#temporalRestrictions &= 14^{numPredPairs} \\ &= 14^{\frac{n \cdot (n-1)}{2}} \end{aligned}$$

Table 5 shows the number of temporal restrictions for different pattern sizes if five predicates are used.

Altogether the number of possible patterns grows very high if the number of used predicates in patterns is large. First of all, many “basic patterns” are created in this case. For each of these patterns many variable unifications are possible (depending on the number of variables in all predicates of the pattern). In the last step for each of the unified patterns many variations of temporal restrictions are possible. This shows that a modification of the algorithm has to be done in order to make it applicable to more complex situations.

## 7. Conclusion and Future Work

The approach presented here allows for learning patterns from sequential symbolic data. As the representation consists of predicates which describe relations between different objects more complex patterns can be learned compared to propositional event sequences. The unsupervised pattern learning algorithm performs a top-down search from the most general to more specific patterns. In the first step frequent basic patterns are generated (similar to standard association rule mining). In the second step variable unification leads to an interconnection of predicates and thus to more coherent patterns. In the third step the patterns are specialized by temporal constraints. Here, Allen’s interval logic is used to interrelate the predicates in the patterns temporarily. An evaluation function was introduced, which takes into account the special properties of our pattern representation. The algorithm was implemented and tested on artificial data sets.

Although the basic algorithm has been implemented already there are many possibilities for future work. Even

though the incremental generation of new patterns just from frequent patterns excludes many unimportant patterns, the complexity of the algorithm is still very high. An interesting research direction is to find out which heuristics could reduce complexity and still lead to good patterns. Another possibility is to prevent the program from the generation of certain patterns by leaving out (temporal) relations which are not possible due to some constraints. These constraints could be identified e.g., by accessing a composition table as introduced by Allen [3] as it was done by Höppner [5]. In some cases it might be also possible to reduce the number of temporal relations. If some of the 13 relations are not needed (or do not appear in the data) they can be omitted in order to reduce complexity.

The creation of temporal restrictions is quite straightforward so far as just one temporal relation is allowed between two predicates. If the semantic is changed to “allowed temporal relations”, more than one temporal relation could be allowed for each predicate pair. This would lead to a more powerful representation of the temporal restrictiveness.

In the current version of the algorithm just frequent patterns are generated. In future versions it would be interesting to create (temporal) association rules, which can be applied for anticipating (possible) future events.

In future work the pattern learning has to be evaluated in some application context with real data in order to evaluate how beneficial such learned patterns can be. This can be done in the context of intelligent agents which should use learned patterns to improve their future behavior, e.g., by avoiding unfavorable or dangerous situations or by reusing partial plans which have been computed in similar situation before to save time.

## 8. Acknowledgements

The content of this paper is a partial result of the ASKOF project which is funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) under grant HE989/6-1. We would like to thank Björn Gottfried, Martin Lorenz, Ingo J. Timm, and Tom Wetjen for interesting discussions on the learning of patterns and complexity calculations, and for their comments how to improve this paper.

## References

- [1] R. Agrawal, T. Imielinski, and A. N. Swami. Mining association rules between sets of items in large databases. In *Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data*, pages 207–216, Washington, D.C., May 1993.
- [2] R. Agrawal and R. Srikant. Fast algorithms for mining association rules. In *Proceedings of the 20th International Conference on Very Large Data Bases, VLDB*, pages 487–499, September 1994.
- [3] J. F. Allen. Maintaining knowledge about temporal intervals. *Communications of the ACM*, 26(11):832–843, November 1983.
- [4] E. T. Bell. Exponential numbers. *The American Mathematical Monthly*, 41(7):411 – 419, August/September 1934.
- [5] F. Höppner. Learning temporal rules from state sequences. In *Proceedings of the IJCAI’01 Workshop on Learning from Temporal and Spatial Data*, pages 25–31, Seattle, USA, 2001.
- [6] K. Koperski and J. Han. Discovery of spatial association rules in geographic information databases. In *Proceedings of the 4th International Symposium on Advances in Spatial Databases, SSD*, pages 47–66, Portland, Maine, 1995.
- [7] D. Malerba and F. A. Lisi. An ILP method for spatial association rule mining. In *Working notes of the First Workshop on Multi-Relational Data Mining*, pages 18–29, Freiburg, Germany, 2001.
- [8] H. Mannila, H. Toivonen, and A. I. Verkamo. Discovery of frequent episodes in event sequences. *Data Mining and Knowledge Discovery*, 1:259–289, 1997.
- [9] J. Mennis and J. W. Liu. Mining association rules in spatio-temporal data. In *Proceedings of the 7th International Conference on GeoComputation*, University of Southampton, UK, 8 - 10 September 2003.
- [10] A. Miene, A. D. Lattner, U. Visser, and O. Herzog. Dynamic-preserving qualitative motion description for intelligent vehicles. In *Proceedings of the IEEE Intelligent Vehicles Symposium (IV ’04)*, pages 642–646, June 14-17 2004.
- [11] A. Miene, U. Visser, and O. Herzog. Recognition and prediction of motion situations based on a qualitative motion description. In D. Polani, B. Browning, A. Bonarini, and K. Yoshida, editors, *RoboCup 2003 – Proceedings of the International Symposium, July 10.-11., 2003, Padua, Italy*, 2003.
- [12] J. Pitman. Some probabilistic aspects of set partitions. *The American Mathematical Monthly*, 104(3):201–209, March 1997.
- [13] P.-N. Tan, V. Kumar, and J. Srivastava. Selecting the right interestingness measure for association patterns. In *Proceedings of the Eight International Conference on Knowledge Discovery and Data Mining (KDD’02)*, pages 32–41, 2002.
- [14] M. J. Zaki, S. Parthasarathy, M. Ogihara, and W. Li. New algorithms for fast discovery of association rules. In *Proceedings of the 3rd International Conference on Knowledge Discovery and Data Mining*, pages 283–296, 1997.