Relational Sequence Learning

Kristian Kersting[1], Luc De Raedt[2], Bernd Gutmann[2], Andreas Karwath[3], and Niels Landwehr[3]

[1] CSAIL, Massachusetts Institute of Technology
32 Vassar Street, Cambridge, MA 02139-4307, USA
kersting@csail.mit.edu
[2] Departement Computerwetenschappen, K.U. Leuven
Celestijnenlaan 200A - bus 2402, B-3001 Heverlee, Belgium
{Luc.DeRaedt,Bernd.Gutmann}@cs.kuleuven.be
[3] Machine Learning Lab, Institute for Computer Science, University of Freiburg
Georges-Koehler Allee, Building 079, 79110 Freiburg, Germany
{landwehr,karwath}@informatik.uni-freiburg.de

Abstract. Sequential behavior and sequence learning is essential to intelligence. Often the elements of sequences exhibit an internal structure that can elegantly be represented using relational atoms. Applying traditional sequential learning techniques to such relational sequences requires either to ignore the internal structure or to put up with a combinatorial explosion in the model complexity. This chapter briefly reviews relational sequence learning and describes methods that have been developed such as data mining techniques, (hidden) Markov models, conditional random fields, dynamic programming and reinforcement learning techniques.

1 Introduction

Sequential behavior is essential to intelligence, and it is a fundamental part of human activities ranging from reasoning to language, and from everyday skills to complex problem solving. In particular, sequence learning is an important component of learning in many task domains such as planning, reasoning, robotics, user modeling, natural language processing, speech recognition, adaptive control, activity recognition, information extraction, and computational biology. Therefore it is not surprising that sequential data has been the subject of active research for last few decades. Learning tasks investigated include prediction, alignment, classification, labeling, and density and policy estimation.

One major dimension along which to differentiate sequential learning techniques is the complexity of the language they employ to describe sequences and models. At one extreme are learning approaches that assume a propositional language. The simplicity of a propositional language allows such methods to represent the model in matrix form: cells typically denote the transition probabilities among symbols. Indeed, matrices are simple and efficient matrix operations can be used. In turn, a matrix form makes it possible to devise efficient algorithms. At the other end of the spectrum, (probabilistic) relational systems

accept descriptions of complex, structured sequence elements and generate relationally structured models. They typically have access to background knowledge and require fewer entity description. This chapter presents several relational sequences learning techniques that build on ideas developed on both sides of the spectrum. They fill an interesting, intermediate position on the expressiveness scale, namely sequences of relational atoms.

The following section briefly reviews sequential learning. After illustrating the inadequacies of propositional languages, the chapter introduces the more complex data model of relational sequences. In the remaining sections, we will then present methods for dealing with relational sequences.

2 Sequential Learning

Consider sequences of UNIX commands. They typically tell a lot about the user herself since users tend to respond in a similar manner to similar situations, leading to repeated sequences of actions.

Example 1. For instance, LaTeX users frequently run EMACS to edit their LaTeX files and afterwards compile the edited file using LaTeX:

$$\texttt{emacs rsl.tex}, \texttt{ls}, \texttt{latex dvips.tex}, \texttt{dvips rsl} \ \ldots \tag{1}$$

The existence of command alias mechanisms in many UNIX command interpreters also supports the idea that users tend to enter many repeated sequences of commands. Thus, UNIX command sequences carry a lot information, which can be used to automatically construct user profiles, which in turn can be used to predict the next command, the identify the current user etc.

In general, sequence learning considers essentially strings (command logs) $s = \mathtt{w}_1, \mathtt{w}_2, \ldots, \mathtt{w}_T$ ($T > 0$) of symbols $\mathtt{w}_i$ (UNIX commands) over an alphabet Σ. With some necessary simplification, we can identify various sequence learning problems. In *sequence prediction* for instances we want to predict elements (commands) of a sequence based on preceding elements (commands), i.e., $\mathtt{w}_{t-k}, \mathtt{w}_{t-k+1}, \ldots, \mathtt{w}_t \rightarrow \mathtt{w}_{t+1}$. In *frequent sequence mining* for instance, we want to compute the (sub)sequences frequently occurring in a set of sequences. In *sequence prediction*, we want to predict elements (commands) of a sequence based on preceding elements (commands), i.e., $\mathtt{w}_{t-k}, \mathtt{w}_{t-k+1}, \ldots, \mathtt{w}_t \rightarrow \mathtt{w}_{t+1}$. In *sequence classification* we want to predict a single class label (user or user type) $\mathtt{c}$ that applies to an entire sequence s, i.e., $s \rightarrow \mathtt{c}$. In *sequence labeling*, we want to assign a (class) labels (shell sessions) $\mathtt{c}_i$ to each sequence element $\mathtt{w}_i$, i.e., $\mathtt{w}_1, \mathtt{w}_2, \ldots, \mathtt{w}_T \rightarrow \mathtt{c}_1, \mathtt{c}_2, \ldots, \mathtt{c}_T$. *Sequential decision making* involves selecting sequences of actions to accomplish a goal or to maximize the future reward function (for instance to optimally organize email folders). In addition, there are also other issues arising out, or along with, these above sequence learning tasks. For example, we may want to segment a sequence, cluster set of sequences, or we may align two or more sequences.

Another dimension along which one can characterized sequential learning is the learning paradigm followed. Learning tasks can be supervised, unsupervised,

or reinforced. In all cases, however, sequence learning methods essentially rely on models for "legitimate" sequences (in the form of production rules, Markov chains, hidden Markov models, or some other form), which can typically be developed from data using grammar learner, expectation-maximization, gradient descent, policy iteration or some other form of machine learning algorithm. All prominent types of models investigated over the last decades, however, essentially share a principal weakness that stems from a lack of expressive power in the language used to described sequences and models.

3 Moving to More Complex Sequences

Prominent sequence learning techniques such as (hidden) Markov models assume atomic representations, which essentially amounts to enumerate all unique configurations. It might then be possible to learn, for example, that state `state234` follows (with high probability) `state654321`. Atomic representations are simple and learning can be implemented using efficient matrix operations. These matrices, however, can become intractably large as they scale quadratically in the size of the language. This problem becomes is even prominent when the sequence element are structured.

In many applications, sequence elements are indeed structured and can elegantly be represented as relational *ground atoms*.

Example 2. Using ground atoms, the UNIX command sequence of Example 1 can be represented as

$$\texttt{emacs}(\texttt{rsl},\texttt{tex}),\texttt{ls},\texttt{latex}(\texttt{rsl},\texttt{tex}),\texttt{dvips}(\texttt{rsl},\texttt{dvi})\ldots$$

Here, `emacs`/2, `ls`/0, `latex`/1, `dvips`/2 are *predicates* (of arity 2, 0 and 1 respectively) that identify relations. Lower-case strings like `rsl`, `tex`, and `dvi` are *constants*. *Ground atoms* are now predicates together with their arguments, for example $\texttt{emacs}(\texttt{rsl},\texttt{tex})$ and `ls`. In principle, symbols can even be described propositionally, i.e., conjunctions of ground atoms. For instance, the conjunction $\texttt{file}(\texttt{f1}),\texttt{name}(\texttt{f1},\texttt{rsl}),\texttt{suffix}(\texttt{f1},\texttt{tex})$ describes that there is a file with name `rsl` and suffix `tex`. Though this representation affords some opportunities for generalization, we must still refer to objects by name such as $\texttt{file}(\texttt{f1})$. This prevents generalization over several entities such as $\texttt{emacs}(\texttt{X},\texttt{tex})$. The *abstract symbol* $\texttt{emacs}(\texttt{X},\texttt{tex})$ is — by definition — a logical atom, i.e., a predicate together with it arguments, where an argument can now be a placeholder $\texttt{X},\texttt{Y},\ldots$ for some constant. It is abstract in that it represents the set of all ground, i.e., variable-free atoms such as $\texttt{emacs}(\texttt{rsl},\texttt{tex})$, $\texttt{emacs}(\texttt{rsl},\texttt{dvi})$, $\texttt{emacs}(\texttt{april},\texttt{tex})$ etc. Moreover, unification allows to share information between subsequent symbols. For example, the (abstract) sub-sequence $\texttt{latex}(\texttt{X},\texttt{tex}),\texttt{dvips}(\texttt{X},\texttt{dvi})$ describes that a user, after compiling a LaTeXfile into a DVI file, turns the DVI into a PostScript file, without stating the name of the file. This is especially important when generalizing patterns across filenames as the objects referred to will typically be different, and the precise identifiers do not matter but the relationships and events they occur in do.

Having specified a more complex language to describe sequences, the next step is to develop sequential learning methods capable of exploiting it. This is what relational sequence learning is about. In the remaining sections, we will review several relational sequence learning and mining methods that have been proven successful in applications. Their underlying idea is to make use of relational abstraction: similar symbols are grouped together by means of logical variables and knowledge is shared across abstract symbols by means of unification. More precisely, we will discuss relational sequence mining, alignment, Markov models, and reinforcement learning in turn.

4 Mining Logical Sequences

Many of the traditional data mining tasks can be phrased as finding the set of patterns $Th(\mathcal{L}, D, q) = \{\phi \in \mathcal{L} | q(\phi, D) \text{ holds } \}$, cf. [21]. Here, $\mathcal{L}$ is the space or language of all possible patterns, D is a set of observations, and q is a predicate or constraint that characterizes the solutions to the data mining task.

The MineSeqLog algorithm of [20] (see also [3]) is a constraint based pattern mining system for logical sequences. The basic component is a frequent pattern miner, which makes the following choices in $Th(\mathcal{L}, D, q)$:

- D is a set of ground logical sequences over an alphabet Σ
- $\mathcal{L}$ consists of the abstract sequences over Σ, (in which variables can occur)
- q is a constraint of the form $freq(\phi, D) \geq t$ expressing that the pattern ϕ must cover at least t of the sequences in D

This formulation makes some simplifications, in that MineSeqLog can also cope with sequences with gaps as well as with other constraints than a minimum frequency threshold, cf. [20].

The key constraint is the minimum frequency threshold. The *frequency*, $freq(\phi, D)$a, of a pattern (in the form of an abstract sequence) ϕ is the number of sequences s in D for which ϕ subsumes s. A sequence $s = \mathtt{w}_1, \mathtt{w}_2, \ldots, \mathtt{w}_T$ is subsumed by a pattern $\phi = \mathtt{p}_1, \ldots, \mathtt{p}_k$ if and only if there exists a substitution θ and natural numbers i, j such that $\mathtt{p}_1\theta = \mathtt{w}_i, \mathtt{p}_2\theta = \mathtt{w}_{i+1}, \ldots, \mathtt{p}_k\theta = \mathtt{w}_j$. For instance, the pattern $\mathtt{latex}(\mathtt{File}, \mathtt{tex}), \mathtt{dvipdf}(\mathtt{File}, \mathtt{dvi})$ subsumes the concrete sequence $\mathtt{cd}(\mathtt{april}), \mathtt{latex}(\mathtt{par}, \mathtt{tex}), \mathtt{dvipdf}(\mathtt{par}, \mathtt{dvi}), \mathtt{lpr}(\mathtt{par}, \mathtt{pdf})$ with substitution $\theta = \{\mathtt{File}/\mathtt{april}\}$. We sometimes will say that ϕ is more general than s, or vice versa, that s is more specific than ϕ. The subsumption relation induces a partial order on the language $\mathcal{L}$, which is used in order to structure the search for frequent patterns. The subsumption ordering can be exploited because the minimum frequency constraint is anti-monotonic. More formally, a constraint q is *anti-monotonic* if and only if $\forall$ sequences x: x subsumes $y \wedge p(y) \rightarrow p(x)$. It is easy to see that this holds for frequency because the frequency can only decrease when refining patterns. The anti-monotonicity property implies that there is a border of maximally specific sequences satisfying the constraint.

The set $Th(\mathcal{L}, D, freq(\phi, D))$ can now be computed by instantiating the traditional level-wise algorithm of [21], which is essentially a breadth-first general-

to-specific search algorithm. To generate more specific patterns from more general ones, a refinement operator ρ is employed. A refinement operator is an operator ρ that maps each sequence s to a set of specializations of it, i.e. $\rho(s) \subseteq \{s' \in \mathcal{L} \mid s \text{ subsumes } s'\}$. Furthermore, to avoid generating the same pattern more than once, the operator should be *optimal*, i.e.

Complete Applying the operator ρ on ϵ, the empty sequence (possibly with repetitions), it is possible to generate all other queries in $\mathcal{L}$. This requirement guarantees that we will not miss any queries that may satisfy the constraints.

Single path Given pattern p, there should exist exactly one sequence of patterns $p_0 = \epsilon, p_1, \ldots, p_T = p$ such that $p_{i+1} \in \rho(p_i)$ for all i. This requirement helps ensuring that no query is generated more than once. i.e. There are no duplicates.

The following operator satisfies these requirements. $\rho(\mathtt{s}_1, \ldots, \mathtt{s}_l)$ is obtained by applying one of the following operations.

Add an atom $\mathtt{s}_{l+1}$ to the right of the query such that $\mathtt{s}_{l+1}$ is an atom whose arguments are different variables not yet occurring in $\mathtt{s}_1, \ldots, \mathtt{s}_l$

Apply a substitution of the form $\theta = \{X/c\}$, where X is a variable, c a constant such that there are no constants occurring to the right of X in $\mathtt{s}_1, \ldots, \mathtt{s}_l$, and all variables in $\mathtt{s}_1, \ldots, \mathtt{s}_l$ are different

Unify two variables X and Y such that X occurs only once, all variables to the right of X occur only once, and X occurs to the right of Y.

This operator can then be integrated in the standard level-wise algorithm for frequent pattern mining. This algorithm is sketched below. It starts from the empty sequence and repeatedly generates candidates (on C_i) to determine afterwards (using F_i) whether they are frequent. To generate candidates the refinement operator ρ is applied. Furthermore, only frequent sequences are refined due to the anti-monotonicity property. This process continues until no further candidates are frequent.

Algorithm 1: Computing the frequent sequences.

$i := 0$; $C_0 := \{\epsilon\}$; $F_0 := \emptyset$
while $C_i \neq \emptyset$ **do**
  $F_i := \{p \in C_i \mid freq(p, D) \geq t\}$
  output F_i
  $C_{i+1} := \{p \mid p \in \rho(p'), p' \in F_i\}$
  $i := i + 1$

The MineSeqLog algorithm as described by [20] cannot only cope with anti-monotonic constraints, but also with monotonic ones, and even conjunctions of the two. A maximum frequency threshold, which is of the form $freq(\phi, D) < f$,

is a monotonic constraint. [20] also report on experiments with MineSeqLog using the Unix-command data set of [8] and constraints of the form $(freq(\phi, ClassA) \geq f_1) \wedge (freq(\phi, ClassB) < f_2)$.

5 Relational Alignments

The need to measure sequence similarity arises in many application domains and often coincides with sequence alignment: the more similar two sequences are, the better they can be aligned. Aligning sequences not only shows how similar sequences are, it also shows where there are differences and correspondences between the sequences. As an example, consider the major application area for sequence alignment in the biological domain. One common approach is, given the amino acid sequence of an unknown protein (query sequence) to scan an existing database of other amino acids sequences (containing proteins with more or less known function) and extract the most similar ones with regard to the query sequence. The result is usually a list, ordered by some score, with the best hits at the top of this list. The common approach for biologists, is now to investigate these top scoring alignments or hits to conclude about the function, shape, or other features of query sequence.

5.1 Sequence Alignment Algorithms

One of the earliest alignment algorithm is that for global alignment by Needleman and Wunsch in 1970 [24]. The algorithm is based on dynamic programming, and is able to find the alignment of two sequences with the maximal overall similarity w.r.t. a given pairwise similarity model. More precisely, the algorithm proceeds as follows: initially, for two sequences of length l and k, a matrix with $l + 1$ columns and $k + 1$ rows is created. The matrix then is filled with the maximum score as follows:

$$M_{i,j} = \max \begin{cases} M_{i-1,j-1} + S_{i,j} & : \text{a match or mismatch} \\ M_{i,j-1} + w & : \text{a gap in the first sequence} \\ M_{i-1,j} + w & : \text{gap in the second sequence} \end{cases} \quad (2)$$

where $S_{i,j}$ is the pairwise similarity of amino acids and w reflects a linear gap (insert or deletion step) penalty. The overall score of the alignment can be found in cell $M_{l,k}$.

In the biological domain, this similarity model $S_{i,j}$ is typically represented by pair-wise similarity or dissimilarity scores of pairs of amino acids. These scores are commonly specified by a so-called similarity matrix, like the PAM [4] or BLOSUM [10] families of substitution matrices. The scores, or costs, associated with a match or mismatch between two amino acids, reflect to some extent the probability that this change in amino acids might have occurred over time of evolution.

The Needleman-Wunsch algorithm attempts to align every element in every sequence. By contrast, local alignments identify regions of similarity within long

1: -	vi(ch2,tex)	ls	latex(ch2,tex)	xdvi(ch2,dvi)	dvipdf(ch2,dvi)	pdfview(ch2,pdf)
2: cd(thesis)	vi(ch1,tex)	bibtex(ch1)	latex(ch1,tex)	xdvi(ch1,dvi)	dvipdf(ch1,dvi)	pdfview(ch1,pdf)
3: -	-	-	-	xdvi(pap2,dvi)	dvipdf(pap2,dvi)	pdfview(pap2,pdf)
4: cd(pap1)	-	-	vi(pap1,tex)	latex(pap1,tex)	dvipdf(pap1,dvi)	pdfview(pap1,pdf)
5: -	vi(rsl,tex)	-	latex(rsl,tex)	dvips(rsl,dvi)	-	-

Table 1. The multiple alignment of five arbitrary Unix command line sequences using gap opening cost 1.5, gap extention cost 0.5, and padding cost 0.25. The '-' denotes a gap in a sequence. Clearly one can see the aligned commands for xdvi, dvipdf, and pdfview. In sequence four, the corresponding vi and latex commands are not properly aligned due to the gap opening costs, as the proper alignment would require two gaps instead of the single one employed here.

sequences that are often widely divergent overall. To calculate the best *local* alignment of two sequences, one often employs the Smith-Waterman local alignment algorithm [31]. The main difference of this algorithm when compared to the Needleman-Wunsch algorithm, is that all negative scores are set to 0.

In general, the alignments resulting from an global or local alignment, show then the more *conserved* regions between two sequences. To enhance the detection of these conserved regions, commonly multiple sequence alignments are constructed. Given a number of sequences belonging to the same class, i.e. in biological terms believed to belong to the same family, fold, or are somehow otherwise related, alignments are constructed by aligning all sequences in one single alignment, a so-called profile. A common approach for the construction of a multiple alignment is a three step approach: First, all pairwise alignments are constructed. Second, using this information as starting point a phylogenetic tree is created as *guiding tree*. Third, using this tree, sequences are joined consecutively into one single alignment according to their similarity. This approach is known as the neighbor joining approach [28].

5.2 Moving towards the Alignment of Relational Sequences

The alignment algorithms discussed in the previous paragraphs assume a given similarity measure $S_{i,j}$. Typically, this similarity measure is a propositional one as the considered sequences consist of propositional symbols. Many sequences occurring in real-world problems, however, can elegantly be represented as relational sequences. A *relational sequence alignment* simply denotes the alignment of sequences of such structured terms.

One attractive way to solve this problem is to use a standard alignment algorithm but to replace the propositional similarity measure $S_{i,j}$ in Eq. (2) by a structured one. In [12] we proposed the use of one of the many distance measures developed within the field of Inductive Logic Programming [23]. As an example, consider one of the most basic measures proposed by Nienhuys-Cheng [25] [4]. It treats ground structured terms as hierarchies, where the top structure is most

[4] For sequences of more complex logical objects such as interpretations and queries, a different, appropriate similarity function could be chosen. We refer to Jan Ramon's PhD Thesis [27] for a nice review of them.

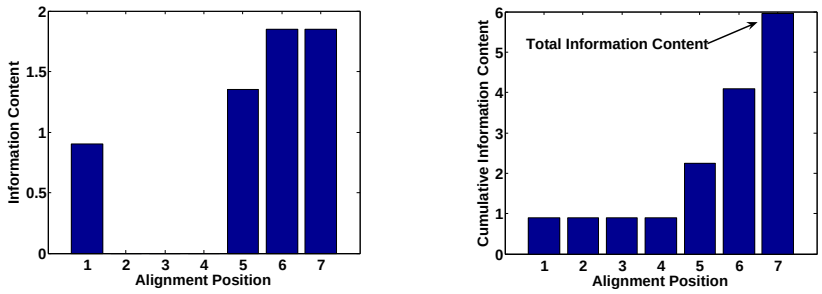

Fig. 1. Information content (IC) for the *Unix command line* example in table 1. The bar graph on the left-hand side shows the IC at each position in the alignment. The bar graph on the right-hand side shows the cumulative IC up to each position in the alignment.

important and the deeper, nested sub-structures are less important. Let $\mathcal{S}$ denote the set of all symbols, then Nienhuys-Cheng distance d is inductively defined as follows:

$$\begin{array}{ll} \forall \mathtt{c}/0 \in \mathcal{S} : & d(\mathtt{c},\mathtt{c}) = 0 \\ \forall \mathtt{p}/u, \mathtt{q}/v \in \mathcal{S} : \mathtt{p}/u \neq \mathtt{q}/v : & d(\mathtt{p}(\mathtt{t}_1,\ldots,\mathtt{t}_u),\mathtt{q}(\mathtt{s}_1,\ldots,\mathtt{s}_v)) = 1 \\ \forall \mathtt{p}/u \in \mathcal{S} : & d(\mathtt{p}(\mathtt{t}_1,\ldots,\mathtt{t}_u),\mathtt{p}(\mathtt{s}_1,\ldots,\mathtt{s}_u)) = \frac{1}{2\mathtt{u}} \sum_{\mathtt{i}=1}^{\mathtt{u}} d(\mathtt{t}_\mathtt{i},\mathtt{s}_\mathtt{i}) \end{array}$$

For different symbols the distance is one; however, when the symbols are the same, the distance linearly decreases with the number of arguments that have different values, and is at most 0.5. The intuition is that longer tuples are more error-prone and that multiple errors in the same tuple are less likely. To solve the corresponding relational alignment problem, one simply sets $S_{i,j} = 1 - d(\mathtt{x}_\mathtt{i}, \mathtt{y}_\mathtt{i})$ in Equation (2).

5.3 Relational Information Content

Now that we have introduced relational sequence alignments, we will investigate how informative they are. Following Gorodkin *et al.* [6], the information content I_i of position i of a relational sequence alignment is $I_i = \sum_{k \in G} I_{ik} = \sum_{k \in G} q_{ik} \log_2 \left(\frac{q_{ik}}{p_k} \right)$, where G is the Herbrand base over the language of the aligned sequences including gaps (denoted as '$-$') and q_{ik} is the fraction of ground atoms k at position i. When k is not a *gap*, we interpret p_k as the *a priori* distribution of the ground atom. Following Gorodkin *et al.*, we set $p_- = 1.0$, since then $q_{i-} \log_2(q_{i-}/p_-)$ is zero for q_{i-} equal to zero or one. For the work

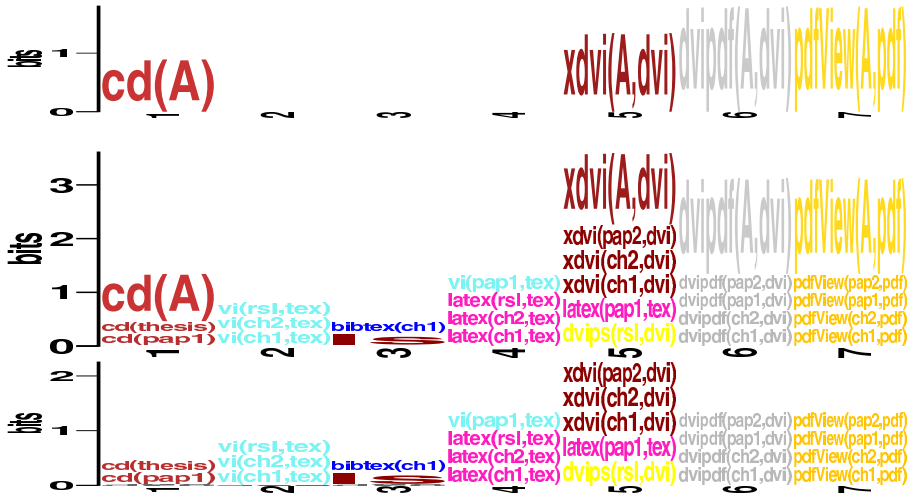

Fig. 2. Sequence logos for the *Unix command line* example in table 1 (from bottom to top: ground, relational, and abstract). For positions 1 as well as positions 5 -7 one can clearly see that the abstract logo contributes substantially towards a conserved region.

reported here, we set $p_k = 1/(|G| - 1)$ when $k \neq -$. The intuition is as follows: *if I_{ik} is negative, we observe fewer copies of ground atom k at position i than expected, and vice versa if I_{ik} is positive, we observe more of it.*

The total information content becomes $I = \sum_{i=1}^{T} I_i$ (where T is the length of the alignment) and can be used to evaluate relational sequence alignments. So far, however, we have defined the information content at the most informative level only, namely the level of ground atoms. Relational sequences exhibit a rich internal structure and, due to that, multiple abstraction levels can be explored: variables allow to make abstraction of specific symbols. To compute the information content at a higher abstraction levels, i.e., of an atom a replacing all covered ground atoms k at position i, we view q_{ia} (resp. p_a) as the sum of q_{ik} (resp. p_k) of the ground atoms k covered by a. Figure 1 shows the (cumulative) information content for our running *Unix command line* example. As prior we use the empirical frequencies over all five sequences.

The information content is a significant concept as it allows to evaluate alignments of and to find common motifs in relational sequences. Moreover, it allows one to represented alignments graphically by so-called *relational sequence logos*.

5.4 Relational Sequence Logos

Reconsider the alignment in Table 1. It consists of several lines of information. This makes it for longer sequences difficult – if not impossible – to read off information such as the general consensus of the sequences, the order of predominance

of the symbols at every position, their relative frequencies, the amount of information present at every position, and significant locations within the sequences. In contrast, the corresponding sequence logo as shown in Figure 2 concentrates all of this into a single graphical representation. In other words, '*a logo says more than a thousand lines alignment*'.

Each position i in a *relational sequence logo* is represented by a stack consisting of the atoms at position i in the corresponding alignment. The height of the stack at position i indicates the information content I_i available. The height h_{ik} of each atom k at position i is proportional to its frequency relative to the expected frequency, i.e., $h_{ik} = \alpha_i \cdot \left(\frac{q_{ik}}{p_k}\right) \cdot I_i$, where α_i is a normalization constant. The atoms are sorted according to their heights. If I_{ik} is negative, a simple bar is shown using the absolute value of the I_{ik}.

Sequence logos at lower abstraction levels can become quite complex. Relational abstraction can be used to straighten them up. Reconsider Figure 2. It also shows the logo at the highest abstraction level, where we considered as symbols the *least general generalization* of all ground atoms over the same predicate at each position in the alignment only. Because the prior probabilities change dramatically, the *abstract logo* looks very different from the ground one. It actually highlights the more conserved region of the sequences at the end (positions 5-7). Both views provide relevant information. *Relational logos* now combine both by putting at each position the individual stack items together and sort them in ascending order of heights.

To summarize, relational sequence logos illustrate that while relational alignments can be quite complex, they exhibit rich internal structures which, if exploited, can lead to new insights not present in flat alignments. For applications to information extraction from MedLine abstracts and to and protein fold description, we refer to [12].

Both *frequent sequence mining* and *sequence alignment* are nonparametric methods in that they do not assume an underlying model. We will now turn to model-based sequence learning methods. More precisely, we will focus on probabilistic sequence models. They are an appealing approach to sequential learning as they take uncertainty explicitly into account.

6 Relational Grams

Relational Grams extend *n-gram* models to sequences of logical atoms. In a nutshell, n-gram models are smoothed Markov chains: they model the probability of a sequence $s = w_1...w_m$ as a mixture of Markov distributions of different orders. For $k \in \mathbb{N}$, a k-th order Markov chain estimates the probability for s as

$$P_k(w_1...w_m) = \prod_{i=1}^{m} P_k(w_i \mid w_{i-k+1}...w_{i-1}) \tag{3}$$

In the most basic case, the conditional probabilities are estimated from a set S of training sequences in terms of "gram" counts, i.e., counts of short patterns of

symbols such as $w_{i-k+1}...w_{i-1}$:

$$P_k(w_i \mid w_{i-k+1} \dots w_{i-1}) = \frac{C(w_{i-k+1} \dots w_i)}{C(w_{i-k+1} \dots w_{i-1})} \tag{4}$$

where $C(w_{i-k+1}...w_{i-1})$ is the number of times $w_{i-k+1}...w_{i-1}$ appeared as a subsequence in any $s \in S$. This is the maximum likelihood estimate for the model described by Equation 3.

In this model, the gram order k defines the trade-off between reliability of probability estimates and discriminatory power of the model. For larger k, many probability estimates will typically be zero due to data sparseness, which can deteriorate model accuracy. n-grams combine models of different order [22], and estimate the conditional probabilities as

$$P(w_i \mid w_{i-n+1} \dots w_{i-1}) = \sum_{k=1}^{n} \alpha_k P_k(w_i \mid w_{i-k+1} \dots w_{i-1}) \tag{5}$$

where the $\alpha_1, ..., \alpha_n$ are suitable weights with $\sum_{k=1}^{n} \alpha_k = 1$, and the distributions $P_k(w_i \mid w_{i-k+1} \dots w_{i-1})$ are estimated according to Equation 4. This effectively smoothes the probability estimates of the higher-order models with the more robust estimates of lower-order models and thereby avoids the data sparseness problem. More advanced smoothing techniques have also been proposed (cf. [22]), but they are beyond the scope of this chapter. Despite their simplicity, n-grams have proven to be a powerful tool for sequence classification and probability estimation.

By generalizing the sequence elements w_i to first-order logical atoms, relational grams (or *r-grams*) inherit the power and simplicity of the n-gram method. However, they go beyond a simple relational upgrade of n-grams in two ways:

1. **Relational Smoothing** In addition to smoothing by shorter histories (as for n-grams), relational grams can smooth probability estimates by relational generalization of grams. For example, the gram $emacs(rsl, tex), latex(rsl, tex)$ could be generalized by shortening it to $emacs(rsl, tex)$, but also by logical abstraction to $emacs(F, tex), latex(F, tex)$. Both generalizations avoid the data sparseness problem by estimating probabilities from a larger sample, however, they represent a different bias. In the second case, we model a pattern indicating that a user first runs *emacs* and then *latex* on the same file, independently of a particular filename.
2. **Abstraction from Identifiers** In fact, there are some arguments in the predicates used to define sequence elements which should never be grounded. Consider, for example, filenames in the Unix user modeling domain described above. File names, in contrast to file extensions, are just names—except for some system files, they are chosen more or less arbitrarily to describe the particular content of a file. Accordingly, it does not make sense to estimate distributions over filenames, especially if we want the induced models to generalize across different users, who typically name their files differently. Rela-

tional grams therefore provide a mechanism to abstract from such identifiers, and define distributions over ground sequences modulo identifier renaming.

More formally, r-gram models can be defined as follows. Let Σ denote a typed relational alphabet, for which the set of types is partitioned into *constants* and *identifiers*. We will also talk about *constant-variables* and *identifier-variables* depending on the variable's type. Let $\hat{\Sigma}$ denote the subset of Σ where no arguments of identifier types are grounded.

Definition 1 (r-gram model). *An* ***r-gram model*** R *of order* n *over an alphabet* Σ *is a set of relational grams*

$$l_n^1 \vee ... \vee l_n^d \leftarrow l_1 ... l_{n-1}$$

where

1. $\forall i : l_1 ... l_{n-1} l_n^i \in \bar{\Sigma}^*$;
2. $\forall i : l_n^i$ *contains no constant-variables;*
3. $\forall i : l_n^i$ *is annotated with probability values* $P_r(l_n^i \mid l_1 ... l_{n-1})$ *such that* $\sum_{i=1}^{d} P_r(l_n^i \mid l_1 ... l_{n-1}) = 1$
4. $\forall i \neq j : l_1 ... l_{n-1} l_n^i \not\preceq l_1 ... l_{n-1} l_n^j$; *i.e. the heads are mutually exclusive. Here, the operator* $\preceq$ *implements subsumption under* object identity *as defined in [29].*

Example 3. The following is an example of an order 2 *relational gram in the Unix user domain.*

$$\left.\begin{array}{l} 0.4\ latex(F, tex) \\ 0.1\ latex(F', tex) \\ 0.1\ emacs(F', tex) \\ \ ... \\ 0.05\ cd(Dir) \end{array}\right\} \leftarrow emacs(F, tex)$$

It states that after editing a file with emacs, a user is more likely to use latex on that file than she is to use latex on a different file or execute another command.

We still need to show how an r-gram model R defines a distribution over relational sequences. We first discuss a basic model by analogy to an unsmoothed n-gram, before extending it to a smoothed one in analogy to Equation 5.

A Basic Model In the basic r-gram model, for any ground sequence $g_1 ... g_{n-1}$ there is exactly one gram $l_n^1 \vee ... \vee l_n^d \leftarrow l_1 ... l_{n-1}$ with $l_1 ... l_{n-1} \preceq_\theta g_1 ... g_{n-1}$. Its body $l_1 ... l_{n-1}$ is the most specific sequence subsuming $g_1 ... g_{n-1}$. According to Equation 3, we start by defining a probability $P_R(g \mid g_1 ... g_{n-1})$ for any ground atom g given a sequence $g_1 ... g_{n-1}$ of ground literals. Let g be a ground literal and consider the above gram subsuming $g_1 ... g_{n-1}$. If there is an $i \in \{1, ..., d\}$ such that $l_1 ... l_{n-1} l_n^i \preceq_\theta g_1 ... g_{n-1} g$ it is unique and we define

$$P_R(g \mid g_1 ... g_{n-1}) := P_r(g \mid g_1 ... g_{n-1}) := P_r(l_n^i \mid l_1 ... l_{n-1})$$

Otherwise, $P_R(g \mid g_1...g_{n-1}) = 0$. From $P_R(g \mid g_1...g_{n-1})$, a sequence probability $P_R(g_1...g_m)$ can be derived as in Equation 3.

In this way, the model assigns a probability value to any ground sequence s over the alphabet Σ. If two sequences are identical up to local identifier renaming, the model will assign them the same probability value. For example, the same probability is assigned to $emacs(chapter1, tex), latex(chapter1, tex)$ and $emacs(chapter2, tex), latex(chapter2, tex)$. We have therefore modeled patterns of object identifiers (the fact that the same file name is used in both commands) without referring to any concrete identifiers. As the model does not distinguish between sequences that are identical up to identifier renaming, the sum of probability estimates over all ground sequences is larger than one. However, the model defines a proper probability distribution over the set of equivalence classes modulo local identifier renaming. More details can be found in [18].

Smoothing r-grams In the basic model, there was exactly one gram $r \in R$ subsuming a given ground subsequence $g_1...g_{n-1}$, namely the most specific one. As for n-grams, the problem with this approach is that there is a large number of such grams and the amount of training data needed to reliably estimate all of their frequencies is prohibitive unless n is very small. The basic idea behind smoothing in r-grams is to generalize grams logically, and mix the resulting distributions, i.e., $P_R(g \mid g_1...g_{n-1}) = \sum_{r \in \hat{R}} \frac{\alpha_r}{\alpha} P_r(g \mid g_1...g_{n-1})$ where $P_r(g \mid g_1...g_{n-1})$ is the probability defined by r as explained above, $\hat{R}$ is the subset of grams in R subsuming $g_1...g_{n-1}$, and α is a normalization constant, i.e. $\alpha = \sum_{r \in \hat{R}} \alpha_r$. The more general r, the more smooth the probability estimate $P_r(g \mid g_1...g_{n-1})$ will be. The actual degree and characteristic of the smoothing is defined by the set of matching r-grams together with their relative weights α_r.

To summarize, r-grams upgrade n-grams to deal with sequences of logical atoms. As n-grams, they combine simple Markov models with powerful smoothing techniques. Furthermore, they allow to abstract from identifiers in the data. As for n-grams, learning r-grams is straightforward, and basically amounts to counting frequencies of first-order patterns in the data. These could be determined efficiently e.g. by a first-order sequential pattern miner such as SeqLog [19]. Furthermore, r-grams need a user-defined *language bias*, which constrains the allowed patterns in terms of types and determines which types are treated as identifiers.

r-grams have been successfully applied to structured sequential problems in Unix user modeling, protein fold prediction, and mobile phone user pattern analysis (see [18]).

7 Logical Hidden Markov Models

In r-grams and Markov models in general, the (structured) states are directly visible, and therefore the transition probabilities among states are the only pa-

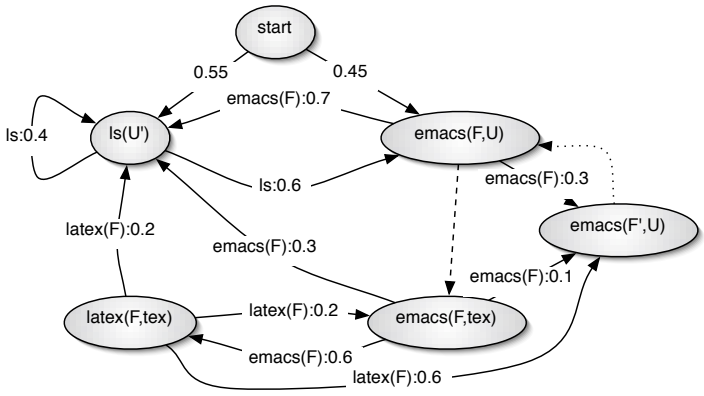

Fig. 3. A logical hidden Markov model. Abstract states are represented by gray nodes. Arrows between nodes denote abstract transitions. The abstract emissions and transition probabilities are associate with the arrows. Dotted arrows denote 'must-follow' links; dashed arrows the 'more-general-than' relation.

rameters. In hidden Markov models [26], the states are not directly observable, but only by means of variables (called observations) influenced by the state.

Definition 2. Abstract transitions *are expressions of the form* $p : \mathtt{H} \xleftarrow{\mathtt{O}} \mathtt{B}$ *where* $p \in [0, 1]$*, and* $\mathtt{H}$, $\mathtt{B}$ *and* $\mathtt{O}$ *are atoms. The atoms* $\mathtt{H}$ *and* $\mathtt{B}$ *are abstract states and* $\mathtt{O}$ *represents an abstract output symbol. All variables are implicitly assumed to be universally quantified, i.e., the scope of variables is a single abstract transition.*

Consider Figure 3. Here, the gray node $\mathtt{emacs}(\mathtt{File}, \mathtt{tex})$ denotes that a LaTeX user edits a file $\mathtt{F}$ using emacs. That the user is indeed a LaTeXuser, however, is not directly observable but only through a sequence of observations such as $\mathtt{emacs}(\mathtt{F})$ and $\mathtt{latex}(\mathtt{F})$ specified in terms of abstract transitions (arrows) such as $c \equiv 0.6 : \mathtt{latex}(\mathtt{File}, \mathtt{tex}) \xleftarrow{\mathtt{emacs(File)}} \mathtt{emacs}(\mathtt{File}, \mathtt{tex})$. Assume now that we are in state $\mathtt{emacs}(\mathtt{rsl}, \mathtt{tex})$, i.e. $\theta_{\mathtt{B}} = \{\mathtt{File}/\mathtt{rsl}\}$. Then c specifies that there is a probability of 0.6 that the next state will be subsumed by $\mathtt{latex}(\mathtt{rsl}, \mathtt{tex})$ and that one of the symbols represented by $\mathtt{emacs}(\mathtt{rsl})$ will be emitted. This was a simple example for an abstract transition because $\theta_{\mathtt{H}}$ and $\theta_{\mathtt{O}}$ were both empty. In general, the resulting state and output symbol sets are not singletons. For instance, for $0.6 : \mathtt{emacs}(\mathtt{File}', \mathtt{U}) \xleftarrow{\mathtt{latex(File)}} \mathtt{latex}(\mathtt{File}, \mathtt{tex})$ the resulting state set is the set of subsumed ground states of $\mathtt{emacs}(\mathtt{File}', \mathtt{U})$ such as $\mathtt{emacs}(\mathtt{rsl}, \mathtt{tex})$, $\mathtt{emacs}(\mathtt{rsl}, \mathtt{dvi})$, $\mathtt{emacs}(\mathtt{lohmm}, \mathtt{tex})$ etc. We therefore need a way to assign probabilities to these possible alternatives.

Definition 3. *The selection distribution* μ *specifies for each abstract state and observation symbol* $\mathtt{A}$ *over the alphabet* Σ *a distribution* $\mu(\cdot \mid \mathtt{A})$ *over all ground atoms subsumed by* $\mathtt{A}$*.*

In our example, assume a selection probability

$$\mu(\texttt{emacs}(\texttt{rsl},\texttt{tex}) \mid \texttt{emacs}(\texttt{File}',\texttt{U})) = 0.4,$$
$$\mu(\texttt{emacs}(\texttt{april},\texttt{tex}) \mid \texttt{emacs}(\texttt{File}',\texttt{U})) = 0.6$$
$$\mu(\texttt{emacs}(\texttt{rsl},\texttt{word}) \mid \texttt{emacs}(\texttt{File}',\texttt{U})) = 0.5,$$
$$\mu(\texttt{emacs}(\texttt{april},\texttt{word}) \mid \texttt{emacs}(\texttt{File}',\texttt{U})) = 0.5$$

Then there would be a probability of $0.4 \times 0.6 = 0.24$ that the next state is $\texttt{emacs}(\texttt{rsl},\texttt{tex})$. Taking μ into account, the meaning of an abstract transition $p : \texttt{H} \xleftarrow{\texttt{O}} \texttt{B}$ can be summarized as follows: *Let* $\texttt{B}\theta_\texttt{B}, \texttt{H}\theta_\texttt{B}\theta_\texttt{H}$ *and* $\texttt{O}\theta_\texttt{B}\theta_\texttt{H}\theta_\texttt{O}$ *be ground atoms. Then the model makes a transition from* $\texttt{O}\theta_\texttt{B}\theta_\texttt{H}\theta_\texttt{O}$ *with probability*

$$p \cdot \mu(\texttt{H}\theta_\texttt{B}\theta_\texttt{H} \mid \texttt{H}\theta_\texttt{B}) \cdot \mu(\texttt{O}\theta_\texttt{B}\theta_\texttt{H}\theta_\texttt{O} \mid \texttt{O}\theta_\texttt{B}\theta_\texttt{H}). \tag{6}$$

To represent μ, any probabilistic representation can - in principle - be used, e.g. a Bayesian network or a Markov chain. In [14], we show how to use a *naïve Bayes* approach to reduce the model complexity.

Thus far the semantics of a single abstract transition has been defined. A logical hidden Markov model usually consists of multiple abstract transitions, which makes things a little bit more complicated. Reconsider Figure 3. Here, *dotted edges* indicate that two abstract states behave in exactly the same way. If we follow a transition to an abstract state with an outgoing dotted edge, we will automatically follow that edge making appropriate unifications. Furthermore, *dashed edges* encode a preference order among abstract states used as conflict resolution strategy. Indeed multiple abstract transitions can match a given ground state. Consider the dashed edge in Figure 3 connecting $\texttt{emacs}(\texttt{File},\texttt{U})$ and $\texttt{emacs}(\texttt{File},\texttt{tex})$ For the state $\texttt{emacs}(\texttt{rsl},\texttt{tex}$ the matching abstract transitions do not sum to 1.0. To resolve this, we only consider the maximally specific transitions (with respect to the body parts $\texttt{B}$) that apply to a state in order to determine the successor states. The rational behind this is that if there exists a substitution θ such that $\texttt{B}_2\theta = \texttt{B}_1$, i.e., $\texttt{B}_2$ subsumes $\texttt{B}_1$, then the first transition can therefore be regarded as more informative than the second one.

Finally, in order to specify a prior distribution over states, we assume a finite set Υ of clauses of the form $p : \texttt{H} \leftarrow \texttt{start}$ using a distinguished $\texttt{start}$ symbol such that p is the probability of the logical hidden Markov model to start in a some ground state.

In [14] it is proven that logical hidden Markov models specify a unique probability measure over sequences of ground atoms over Σ. Moreover all algorithms for hidden Markov models such as the forward, the Viterbi and the Baum-Welch algorithms carry over to the relational case. Thus they can be used for sequence prediction, sequence classification and sequence labeling tasks. Here, we would like to exemplify the practical relevance of logical Hidden Markov models on two bioinformatics domains [16, 14]: Protein fold classification and mRNA signal structure detection.

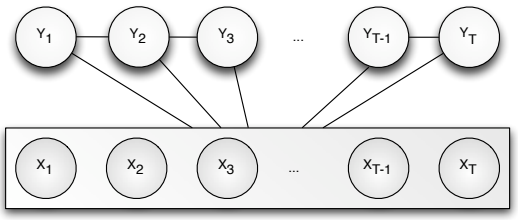

Fig. 4. Graphical representation of linear-chain CRF.

Protein fold classification is concerned with how proteins fold in nature, i.e., their three-dimensional structures. More precisely, given given a protein of unknown fold, assign it to the best matching class of proteins, which were grouped together according to the similarity of their folds. This is an important problem as the biological functions of proteins depend on the way they fold. As already shown in the section on relational alignments, the secondary structure of proteins can elegantly be represented as logical sequences. Here, however, we used a more fine grained discretization of lengths of helices and strands. On a data sets of 816 proteins taken from the ASTRAL dataset version 1.65 consisting of 22210 ground atoms, the 10-fold cross-validated accuracy was 76%. This is in a similar range as Turcotte *et al.*'s [34] 75% accuracy for a similar task. More importantly, the logical hidden Markov models were by an order of magnitude smaller than the number of the equivalent hidden Markov models (120 vs. approx. 62000).

The secondary structure of mRNA contains special subsequences called signal structures that are responsible for special biological functions, such as RNA-protein interactions and cellular transport. In contrast to the secondary structure of proteins, however, it does not form linear chains but trees and hence cannot be represent using ordinary hidden Markov models. We performed leave-one-out cross-validation experiments on a similar data set as used in [11]. The logical hidden Markov models achieved an accuracy of 99% which is well in the range of Horvath *et al.*'s relational instance-based learning approach.

8 Relational Conditional Random Fields

(Logical) HMMs model a sequence X by assuming that there is an underlying sequence of states Y drawn from a finite set of states S. To model the joint distribution $P(X, Y)$ tractably, HMMs make two independency assumptions: each state depends only on its immediate predecessor and each observation/input $\mathtt{x}_j$ depends only on the current state $\mathtt{y}_j$. The downside of these assumptions is that they make it relatively cumbersome to model arbitrary dependencies in the input space, i.e., in the space of X.

For the sequence labeling task, conditional random fields [17] (CRFs) are an alternative to (logical) HMMs that makes it relatively easy to model arbitrary dependencies in the input space. They have become popular in language pro-

cessing, computer vision, and information extraction. They have outperformed HMMs on language processing tasks such as information extraction and shallow parsing. CRFs are undirected graphical models that represent the conditional probability distribution $P(Y|X)$. Instead of the generatively trained (Lo)HMM, the discriminatively trained CRF is designed to handle non-independent input features, which can be beneficial in complex domains.

When used for sequences, the graph structure of a CRF is a first-order chain as shown in Figure 4. Normalization by the $Z(X)$ ensures that the defined function returns a probability:

$$P(Y|X) = \frac{1}{Z(X)} \exp \sum_{t=1}^{T} \Psi_t(y_t, X) + \Psi_{t-1,t}(y_{t-1}, y_t, X). \tag{7}$$

In difference to a Markov Random Field, both the normalization factor $Z(X)$ and the potential functions Ψ are conditioned on the input nodes X. For the sequential learning setting, the potentials are typically represented as a linear combination of feature functions $\{f_k\}$, which are given and fixed:

$$\Psi(y_t, X) = \sum \alpha_k g_k(y_t, X) \text{ and } \Psi(y_{t-1}, y_t, X) = \sum \beta_k f_k(y_{t-1}, y_t, X). \tag{8}$$

The model parameters are then a set of real-valued weights α_k, β_k, one weight for each feature. In linear-chain CRFs (see figure 4), a first-order Markov assumption is made on the hidden variables. In this case, there are features for each label transition. Feature functions can be arbitrary such as a binary test that has value 1 if and only if y_{t-1} has the label a.

So far, CRFs have mainly been applied on propositional input sequences. In the following we will show how to lift them to the relational sequences case. A more detailed description can be found in [9].

8.1 TildeCRF

The only parts of a CRF which access the input sequence are the potential functions. Therefore, CRFs can easily be lifted to the relational sequences case by representing the potential function F as a sum of relational regression trees learned by a relational regression tree learner such as Tilde [2]. Each regression tree stands for a gradient and the sum of all for the potential function. We adapted Dietterich *et al.*'s Gradient Tree Boosting approach, called TreeCRF, to learn the trees. Following their notation, we define $F^{y_t}(y_{t-1}, X) = \Psi(y_t, X) + \Psi(y_{t-1}, y_t, X)$. We do not make assumptions on F, it can be any function not only a linear combination of features.

The gradient of $\frac{\partial \log P(Y|X)}{\partial F^v(u, w_d(X))}$ can be evaluated quite easily for every training sequence and position:

$$\frac{\partial \log P(Y|X)}{\partial F^v(u, w_d(X))} = I(y_{d-1} \prec u, y_d \prec v) - P(y_{d-1} \prec u, y_d \prec v | w_d(X)) \tag{9}$$

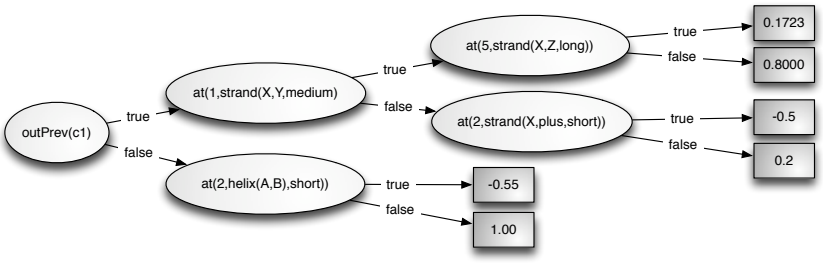

Fig. 5. A relational regression tree; used in TildeCRF to represent the potential function. The inner nodes are logical tests..

where I is the identity function, the symbol $\prec$ denotes that u θ-subsumes y, and $P(y_{d-1} \prec u, y_d \prec v | w_d(X))$ is the probability that class labels u, v fit the class labels at positions $d, d-1$. It is calculated as shown in GenExamples in Alg. 2.

By evaluating the gradient at every known position in our training data and fitting a regression model to this values, we get an approximation of the expectation of the gradient. In order to simplify the derivation of the gradient and afterwards the evaluation, we do not use the complete input X but a window $w_d(X) = x_{d-s}, \ldots, x_d, \ldots, x_{d+s}$, where s is a fixed window size. This is exactly the learning setting of Tilde: each window, i.e., each regression example is a (weighted) set of ground atoms.

All the rest of Dietterich *al.*'s original approach, called TreeCRF, remains unchanged. That is, we can use the forward-backward algorithm as proposed by [5] to compute $Z(X)$. The forward recursion is defined as $\alpha(k, 1) = \exp F^k(\perp, w_1(X))$ and $\alpha(k, t) = \sum_{k' \in K} \left[\exp F^k(k', w_t(X))\right] \cdot \alpha(k', t-1)$. The backward recursion is defined as $\beta(k, T) = 1$ and $\beta(k, t) = \sum_{k' \in K} \left[\exp F^{k'}(k, W_{t+1}(X))\right] \cdot \beta(k', t+1)$.

8.2 Making Predictions

There are several ways for getting a classifier from a trained CRF. We can predict the output sequence Y with the highest probability: $H(X) = \arg\max_Y P(Y|X)$. The Viterbi algorithm [26] can be used for this. Another option is to predict every atom y_t in the output sequence individually. This makes sense when we want to maximize the number of correctly tagged input atoms: $H_t(X) = \arg\max_{k \in K} P(y_t = k|X)$. Finally, one can also use a CRF for sequence classification, i.e., to predict a single label for the entire sequence. To do so, we can simply make a kind of majority vote. That is, we first predict $H(X)$. Next, we count the number of times each class atom was predicted, i.e., $\mathit{count}(c, Y) := |\{i \in \{1, \ldots, T\} \mid y_i = c\}|$. Then, the sequence X is assigned to class c with probability $P(c|X) = T^{-1} \cdot \mathit{count}(c, H(X))$.

We employed TildeCRF to the protein fold classification problem. We used the same subset on the subset of the SCOP database [7] which was mention in

Algorithm 2: Gradient Tree Boosting.

Function TreeCRF($Data, L$)
begin
  for $1 \leq m \leq M$ **do**
    for $1 \leq k \leq K$ **do**
      $Sk :=$GenExamples$(k, Data, Pot_{m-1})$
      $\Delta_m(k) :=$FitRelRegressTree$(S(k), L)$ $F_m^k := F_{m-1}^k + \Delta_m(k)$
  Return Pot_M
end
Function GenExamples($k, Data, Pot_m$)
begin
  $S := \emptyset$
  for $(X_i, Y_i) \in Data$ **do**
    $\big(\alpha, \beta, Z(X_i)\big) :=$ ForwardBackward(X_i, T, K)
    for $1 \leq t \leq T_i$ **do**
      for $1 \leq k' \leq K$ **do**
        `/* Compute value of gradient at position` t `for class label` k `*/`
        $P(y_{t-1} = k', y_t = k|X_i) :=$
        $\dfrac{\alpha(k', t-1) \cdot \exp(F_m^k(k', w_t(X))) \cdot \beta(k, t)}{Z(X_i)}$
        $\Delta(k, k', t) := I(y_{t-1} \prec k', y_t \prec k) - P(y_{t-1} \prec k', y_t \prec k|X_i)$
        `/* add example to set of regression examples */`
        $S := S \cup \{((w_t(X_i), k'), \Delta(k, k', t))\}$
  Return S
end

7. We have done a 10-fold cross-validation and received an overall accuracy of 92.62% which is significantly better than LoHMMs.

To summarize, the previous sections have shown that it is indeed possible to lift many prominent sequence learning techniques to the relational case. Before concluding, we will show that this also holds for relational sequential decision making, i.e., for relational sequence generation through actions.

9 Relational Sequential Decision Making

Animals are able to learn appropriate actions in response to particular stimuli on the basis of associated rewards or punishment is a focus of behavioral psychology. In the machine learning community, learning about stimuli or actions solely on the basis of the rewards and punishments associated with them is called reinforcement learning [32]. It is the problem faced by an agent that acts in

an environment and occasionally receives some reward based on the state the agent is in and the action(s) the agent took. The agent's learning task is to find a policy for action selection that maximizes its reward over the long term. This task requires not only choosing those actions that are associated with high rewards in the current state but also looking ahead by choosing actions that will lead the agent to more lucrative parts of the state space. Thus, in contrast to supervised sequence learning tasks such as sequence labeling, reinforcement learning is minimally supervised because agents are not told explicitly the actions to take in particular situations, but must work this out for themselves on the basis of the rewards they receive.

9.1 Markov Decision Processes

Consider an agent acting in the *blocks world* [30]. The domain consists of a surface, called the *floor*, on which there are *blocks*. Blocks may be on the floor or on top of other blocks. They are said to pile up in stacks, each of which is on the floor. Valid relations are $\texttt{on}(\texttt{X},\texttt{Y})$, i.e., block `X` is on `Y`, and $\texttt{cl}(\texttt{Z})$, i.e., block `Z` is *clear*. At each time, the agent can move a clear (and movable) block `X` onto another clear block `Y`. The $\texttt{move}(X,Y,Z)$ action is probabilistic, i.e., it may not always succeed. For instance, with probability p_1 the action succeeds, i.e. `X` will be on top of `Y`. With probability $1-p_1$, however, the action fails. More precisely, with probability p_2 the block `X` remains at its current position, and with probability p_3 (with $p_1+p_2+p_3=1$) it falls on some clear block `Z`.

A natural formalism to treat the utilities and uncertainties of the blocks world are Markov decision processes. A Markov decision process (MDP) is a tuple $\mathbf{M}=(S,A,\mathbf{T},\lambda)$. Here, S is a set of system states such as

$$z \equiv \texttt{cl}(\texttt{a}),\texttt{on}(\texttt{a},\texttt{b}),\texttt{on}(\texttt{b},\texttt{floor}),\texttt{block}(\texttt{a}),\texttt{block}(\texttt{b})$$

describing the blocks world consisting of two blocks `a` and `b` where `a` is on top of `b`. The agent has available a finite set of actions $A(z) \subseteq A$ for each state $z \in S$, which cause stochastic state transitions, for instance, $\texttt{move}(\texttt{a},\texttt{floor})$ moving `a` on the `floor`. For each $z, z' \in S$ and $a \in A(z)$ there is a transition T in $\mathbf{T}$, i.e., $z' \xleftarrow{p:r:a} z$. The transition denotes that with probability $P(z,a,z') := p$ action a causes a transition to state z' when executed in state z. For instance $z' \equiv \texttt{cl}(\texttt{a}),\texttt{cl}(\texttt{b}),\texttt{on}(\texttt{a},\texttt{floor}),\texttt{on}(\texttt{b},\texttt{floor}),\texttt{block}(\texttt{a}),\texttt{block}(\texttt{b})$. For each $z \in S$ and $a \in A(z)$ it holds $\sum_{z' \in S} P(z,a,z') = 1$. The agent gains a reward when entering a state, denoted as $R(z) := r$. In the blocks world we could have $R(z') = 10$.

A solution of a (relational) Markov decision process is a policy $\pi : S \mapsto A$ mapping state to actions. Essentially policy can be viewed as sets of expressions of the form $a \leftarrow z$ for each $z \in S$ where $a \in A(z)$ such as $\texttt{move}(\texttt{a},\texttt{floor}) \leftarrow \texttt{cl}(\texttt{a}),\texttt{on}(\texttt{a},\texttt{b}),\texttt{on}(\texttt{b},\texttt{floor}),\texttt{blcok}(\texttt{a}),\texttt{block}(\texttt{b})$. It denotes a particular course of actions to be adopted by an agent, with $\pi(z) := a$ being the action to be executed whenever the agent is in state z.

Assuming that the sequence of rewards after step t is $r_{t+1}, r_{t+2}, r_{t+3}, \ldots$, the agent's goal now is to find a policy that maximizes the expected reward

$E[R]$ for each step t. Typically, future rewards are discounted by $0 \leq \lambda < 1$ so that the expected return basically becomes $\sum_{k=0}^{\infty} \lambda^k \cdot r_{t+k+1}$. To achieve this, most techniques employ value functions. More precisely, given some MDP $M = \langle S, A, T, R \rangle$, a policy π for M, and a *discount factor* $\gamma \in [0, 1]$, the *state value function* $V^\pi : S \to \mathbb{R}$ represents the value of being in a state following policy π with respect to the expected reward. In other words, the value $V^\pi(z)$ of a state z is the expected return starting from that state, which depends on the agent's policy π. A policy π' is better than or equal to another policy π, $\pi' \geq \pi$, if and only if $\forall s \in S \ : \ V^{\pi'}(s) \geq V^\pi(s)$. Thus, a policy π^* is optimal, i.e., it maximizes the expected return for all states if $\pi^* \geq \pi$ for all π'. Optimal value functions are denoted V^*. Bellman's [1] *optimality equation* states: $V^*(s) = \max_a \sum_{s'} T(s, a, s')[R(s, a, s') + \gamma V^*(s')]$. From this equation, basically all model-based and model-free methods for solving MDPs can be derived.

9.2 Abstract Policies

A policy over ground states is propositional in the sense that it specifies for each ground state separately which action to execute. In turn, specifying such policies for Markov decision programs with large state spaces is cumbersome and learning them will require much effort. This motivates the introduction of *abstract policies*.

An abstract policies $\boldsymbol{\pi}$ intentionally specify the action to take for sets of states, i.e., for an abstract state.

Definition 4. *An abstract policy* $\boldsymbol{\pi}$ *over* Σ *is a finite set of decision rules of the form* $\mathtt{a} \leftarrow \mathtt{L}$*, where* $\mathtt{a}$ *is an abstract action and* $\mathtt{L}$ *is an abstract state. We assume* $\mathtt{a}$ *to be applicable in* $\mathtt{L}$*, i.e.,* $\mathrm{vars}(a) \subseteq vars(L)$.

The meaning of a single decision rule $a \leftarrow L$ is as follows: *If the agent is in a state Z such that $a \leq_\theta L$, then the agent performs action $a\theta$ with probability $1/|\theta|$, i.e., uniformly with respect to number of possible instantiations of action a in Z.* Usually, however, $\boldsymbol{\pi}$ consists of multiple decision rules. We assume a total order $\prec^{\boldsymbol{\pi}}$ among the decision rules in $\boldsymbol{\pi}$ and use the first matching decision rule such as in Prolog.

Consider the following *unstack-stack* abstract policy:

$\langle 1 \rangle$ $\mathtt{move(A, floor, B)} \leftarrow \mathtt{on(A, B), on(C, D), on(E, floor), cl(A), cl(C), cl(E)}.$
$\langle 2 \rangle$ $\mathtt{move(A, floor, B)} \leftarrow \mathtt{on(A, B), on(C, D), cl(A), cl(C)}.$
$\langle 3 \rangle$ $\mathtt{move(E, A, floor)} \leftarrow \mathtt{on(A, B), on(E, floor), cl(A), cl(E)}.$
$\langle 4 \rangle$ $\mathtt{move(A, B, floor)} \leftarrow \mathtt{cl(A), cl(B)}.$
$\langle 5 \rangle$ $\mathtt{stop} \leftarrow \mathtt{on(A, B), cl(A)}.$

where the `start` action adds the `absorbing` propositions, i.e., it encodes that we enter an absorbing state [5]. For instance in state z (see before), only decision rule $\langle 3 \rangle$ fires.

[5] For ease of exposition, we have omitted the `absorbing` state in front and statements that variables refer to different blocks.

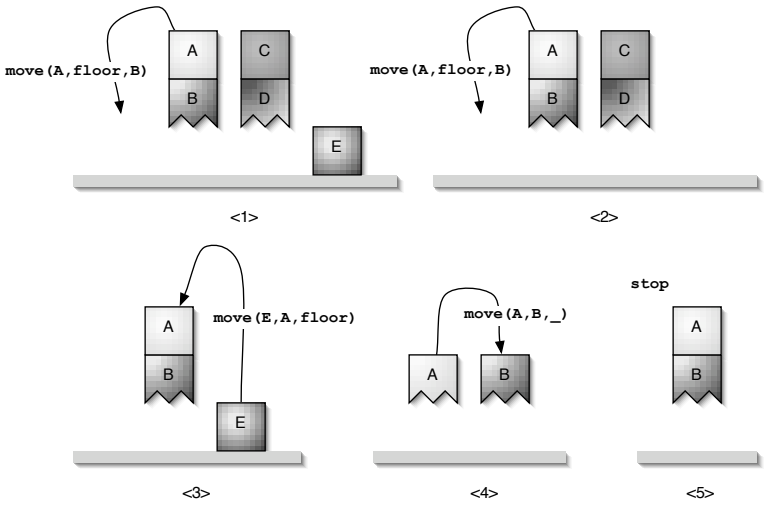

Fig. 6. The decision rules of the *unstack-stack* policy. In the figure, the decision rules are ordered from left to right, i.e., a rule fires only if no rule further to the left fires.

The policy, which is graphically depicted in Figure 6 , is interesting for several reasons. First, it is close to the *unstack-stack* strategy, which is well known in the planning community [30]. Basically, the strategy amounts to first putting all blocks on the table and then building the goal state by stacking all blocks from the floor onto one single stack. No block is moved more than twice. Second, it perfectly generalizes to all other blocks worlds, no matter how many blocks there are. Finally, it cannot be learned in a propositional setting because here the optimal, propositional policy would encode the number of states and the optimal number of moves.

9.3 Relational Temporal Difference Learning

The crucial question for (relational) Markov decision programs and for relational reinforcement learning is how one can learn abstract policies? Almost all relational MDP solvers and reinforcement learning systems follow the so called *generalized relational policy iteration* scheme. It consists of three interacting processes: *policy evaluation*, *policy improvement*, and *policy refinement*. Here, evaluating a policy refers to computing a performance measure of the current policy; policy improvement refers to computing a new policy based on the current value function; and *policy refinement* makes small modifications to an abstract policy such as adding rules.

Here, we will focus on model-free approaches, i.e., we do not assume any model of the world. For a model-based approach, we refer to [15]. Moreover, we will focus on the *relational evaluation problem*, which considers how to compute

Algorithm 3: Relational TD(0) where α is the learning rate and $\widehat{V}(\mathbb{L})$ is the approximation of $V(\mathbb{L})$.

Let $\boldsymbol{\pi}$ be an abstract policy over abstract states $\mathbb{L}$
Initialize $\widehat{V}_0(L)$ arbitrarily for each L in $\mathbb{L}$
repeat
  Pick a ground state $\mathtt{Z}$ of the underlying (relational) MDP M
  repeat
    Choose action $\mathtt{a}$ in $\mathtt{Z}$ based on $\boldsymbol{\pi}$, i.e., (1) select first decision rule $\mathtt{a} \leftarrow \mathtt{L}$ in $\boldsymbol{\pi}$ that matches according to $\prec^{\boldsymbol{\pi}}$, (2) select $\mathtt{a}\theta$ uniformally among induced ground actions
    Take $\mathtt{a}\theta$, observe immediate reward r and successor state $\mathtt{Z}'$, i.e., (1) select with probability p_i the i-th outcome of $\mathtt{a}\theta$, (2) compute $\mathtt{Z}'$ as $[\mathtt{b} \setminus \mathtt{B}\theta] \cup \mathtt{H}_\mathtt{i}\theta$
    Let $\mathtt{L}'$ in $\mathbb{L}$ be the abstract state first matching $\mathtt{Z}'$ according to $\prec^{\boldsymbol{\pi}}$
    $\widehat{V}(L) := \widehat{V}(L) + \alpha \cdot (r + \lambda \cdot \widehat{V}(L') - \widehat{V}(L))$
    Set $\mathtt{Z} := \mathtt{Z}'$
  until $\mathtt{Z}$ *is terminal, i.e., absorbing*
until *converged or some maximal number of episodes exceeded*

the state-value function $V^{\boldsymbol{\pi}}$ for an arbitrary abstract policy $\boldsymbol{\pi}$: **Given** an abstract policy $\boldsymbol{\pi}$, **find** the state-value function $V^{\boldsymbol{\pi}}$ from experiences $\langle S_t, a_t, S_{t+1}, r_t \rangle$ only, where action a_t leads from state S_t to state S_{t+1} receiving reward r_t.

The basic idea is to define the value of an abstract state $\mathtt{L}_i$ (i.e., a body of a decision rule) to be the average expected value for all the states subsumed by that state. This is a good model because if we examine each state subsumed, we make contradictory observations of rewards and transition probabilities. The best model is the average of these observations given no prior knowledge of the model. For ease of explanation, we will focus on a $TD(0)$ approach, see e.g. [32]. Results for general $TD(\lambda)$ can be obtained by applying Tsitsiklis and van Roy's [33] results.

Relational $TD(0)$ sketches the resulting approach. Given some experience following an abstract policy $\boldsymbol{\pi}$, RTD(0) updates its estimate $\widehat{V}$ of V. If the estimated is not changing considerably, the algorithm stops. If an absorbing state is reached, an episode ends and a new "starting" state is selected. If a nonabsorbing state is visited, then it updates its estimate based on what happens after that visit. Instead of updating the estimate at the level of states, $RTD(0)$ updates its estimate at the abstract states of $\boldsymbol{\pi}$ only

RTD(0) can be proven to converge, see e.g. [13]. Figure 7 shows the performance of RTD(0) when evaluating the *unstack-stack* abstract policy (see above). We randomly generated 100 blocks world states for 6 blocks, for 8 blocks, and for 10 blocks using the procedure described by [30]. This set of 300 states constituted the set *Start* of starting states in all experiments. Note that for 10 blocks a traditional MDP would have to represent $58,941,091$ states of which $3,628,800$ are goal states. The result of each experiment is an average of five runs of 5000 episodes, where for each new episode we randomly selected one state from *Start*

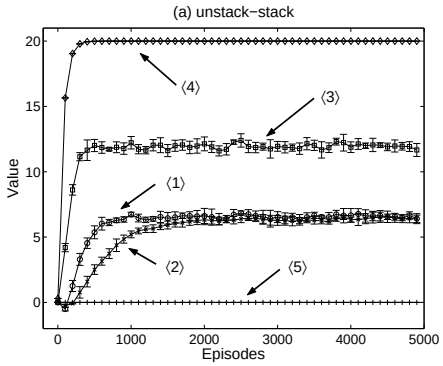

Fig. 7. Relational TD(0)'s learning curves on the evaluation problem for the *unstack-stack* policy. The predicted values are shown as a function of the number of episodes. These data are averages over 5 runs; the error bars show the standard deviations.

as starting state. For each run, the value function was initialized to zero. Furthermore, we used a discount factor λ of 0.9 and a learning rate α of 0.015. The learning curves show that the values of the abstract states converged and, hence, $RTD(0)$ converged. Note that the value of abstract state $\langle 5 \rangle$ remained 0. The reason for this is that, by accident, no state with all blocks on the floor was in *Start*. Furthermore, the values converged to similar values in all runs. The values basically reflect the nature of the policy. It is better to have a single stack than multiple ones.

10 Conclusions

Relational sequences learning problems arise in many applications. This chapter has attempted to describe the relational sequence learning tasks and review some of the leading methods for solving it. In contrast to propositional sequence learning, relational sequence learning assumes the elements of the sequences to be structured in terms of (ground) atoms. This in turn can allow to compress the sequences models tremendously by employing relational abstraction through variables and unification. Our long-term goal should be to develop methods for off-the-shelf relational sequence models. Although we are still some distance from this goal, substantial progress has already been made, and we can look forward to more exciting work in the near future.

Acknowledgments

This work was supported by the European Union, contract number FP6-508861, Applications of Probabilistic Inductive Logic Programming II.

References

1. D. P. Bellman. *Dynamic Programming*. Princeton University Press, 1957.
2. H. Blockeel and L. De Raedt. Top-down Induction of First-order Logical Decision Trees. *Artificial Intelligence*, 101(1–2):285–297, 1998.
3. Maurice Bruynooghe, Luc De Raedt, Sau Dan Lee, and Remko Troncon. Mining logical sequences. Technical report, Department of Computer Science, Katholieke Universiteit Leuven, 2007. Forthcoming.
4. M. O. Dayhoff, R. M. Schwartz, and B. C. Orcutt. A model of evolutionary change in proteins. In M. O. Dayhoff, editor, *Atlas of Protein Sequence and Structure*, volume 5, chapter 22, pages 345–352. Nat. Biomedical Research Foundation, 1978.
5. T. Dietterich, A. Ashenfelter, and Y. Bulatov. Training conditional random fields via gradient tree boosting. In *Proc. 21st International Conf. on Machine Learning*, pages 217–224. ACM, 2004.
6. J. Gorodkin, L. J. Heyer, S. Brunak, and G. D. Stormo. Displaying the information contents of structural RNA alignments: the structure logos. *CABIOS*, 13(6):583–586, 1997.
7. J. Gough, K. Karplus, R. Hughey, and C. Chothia. Assignment of homology to genome sequences using a library of hidden markov models that represent all proteins of known structure. *JMB*, 313(4):903–919, 2001.
8. Saul Greenberg. Using unix: Collected traces of 168 users. Research Report 88/333/45, Department of Computer Science, University of Calgary, Calgary, Canada., 1988.
9. B. Gutmann and K. Kersting. Tildecrf: Conditional random fields for logical sequences. In J. Fürnkranz, T. Scheffer, and M. Spiliopoulou, editors, *Proceedings of the 15th European Conference on Machine Learning (ECML-2006)*, volume 4212 of *LNAI (Lecture Notes in Artificial Intelligence)*, pages 174–185, Berlin, Germany, September 2006. Springer.
10. S. Henikoff and J. G. Henikoff. Amino acid substitution matrices from protein blocks. *Proc. Natl Acad. Sci.*, 89:10915–10919, 1992.
11. T. Horváth, S. Wrobel, and U. Bohnebeck. Relational Instance-Based learning with Lists and Terms. *Machine Learning Journal*, 43(1/2):53–80, 2001.
12. A. Karwath and K. Kersting. Relational sequences alignments and logos. In *Proc. of the 16. Int. Conf. on Inductive Logic Programming (ILP-06)*, pages ??–??, 2007.
13. K. Kersting and L. De Raedt. Logical Markov Decision Programs and the Convergence of Logical TD(λ). In A. Srinivasan, R. King, and R. Camacho, editors, *Proceedings of the Fourteenth International Conference on Inductive Logic Programming (ILP-04*, number 3194 in LNCS, pages 180–197, Porto, Portugal, September 6-8 2004. Springer.
14. K. Kersting, L. De Raedt, and T. Raiko. Logial Hidden Markov Models. *Journal of Artificial Intelligence Research (JAIR)*, 25:425–456, 2006.
15. K. Kersting, M. Van Otterlo, and L. De Raedt. Bellman goes Relational. In R. Greiner and D. Schuurmans, editors, *Proceedings of the Twenty-First International Conference on Machine Learning (ICML-04)*, pages 465–472, Banff, Alberta, Canada, July 4–8 2004.
16. Kristian Kersting, Tapani Raiko, Stefan Kramer, and Luc De Raedt. Towards discovering structural signatures of protein folds based on logical hidden markov models. In *Proceedings of the Pacific Symposium on Biocomputing (PSB-2003)*, pages 192–203, 2003.

17. John Lafferty, Andrew McCallum, and Fernando Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Proc. 18th International Conf. on Machine Learning*, pages 282–289. Morgan Kaufmann, San Francisco, CA, 2001.
18. Niels Landwehr and Luc De Raedt. r-grams: Relational Grams. In *Proceedings of the Twentieth Joint International Conference on Artificial Intelligence (IJCAI-07)*. AAAI Press, 2007.
19. Sau Dan Lee and Luc De Raedt. Mining Logical Sequences Using SeqLog. In *Database technology for data mining*, volume 2682 of *Lecture Notes in Computer Science*. Springer, 2003.
20. Sau Dan Lee and Luc De Raedt. Constraint based mining of first order sequences in seqlog. In Rosa Meo, Pier Luca Lanzi, and Mika Klemettinen, editors, *Database Support for Data Mining Applications*, volume 2682 of *Lecture Notes in Computer Science*, pages 154–173, 2004.
21. Heikki Mannila and Hannu Toivonen. Levelwise search and borders of theories in knowledge discovery. *Data Mining and Knowledge Discovery*, 1(3):241–258, 1997.
22. C. H. Manning and H. Schütze. *Foundations of Statistical Natural Language Processing*. The MIT Press, 1999.
23. S. H Muggleton and L. De Raedt. Inductive Logic Programming: Theory and Methods. *Journal of Logic Programming*, 19(20):629–679, 1994.
24. S. Needleman and C. Wunsch. A general method applicable to the search for similarities in the amino acid sequence of two proteins. *J. Mol. Bio.*, 48(3):443–453, 1970.
25. S.-H. Nienhuys-Cheng. Distance between Herbrand interpretations: A measure for approximations to a target concept. In *Proc. of the 8. Int. Conf. on Inductive Logic Programming (ILP-97)*, pages 250–260, 1997.
26. L. R. Rabiner. A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition. *Proceedings of the IEEE*, 77(2):257–286, 1989.
27. J. Ramon. *Clustering and instance based learning in first order logic*. PhD thesis, Department of Computer Science, K.U. Leuven, Leuven, Belgium, October 2002.
28. N. Saitou and M. Nei. The neighbor-joining method: a new method for reconstructing phylogenetic trees. *Mol. Evol. Biol*, 4(4):406–425, 1987.
29. Giovanni Semeraro, Floriana Esposito, and Donato Malerba. Ideal Refinement of Datalog Programs. In *Proceedings of the 5th Intternational Workshop on Logic Programming Synthesis and Transformation*, 1995.
30. J. Slaney and S. Thiébaux. Blocks World revisited. *Artificial Intelligence Journal*, 125:119–153, 2001.
31. T. F. Smith and M. S. Waterman. Identification of common molecular subsequences. *Journal of Molecular Biology*, 147:195–197, 1981.
32. R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction*. The MIT Press, 1998.
33. J. N. Tsitsiklis and B. Van Roy. An analysis of temporal-difference learning with function approximation. *IEEE Transactions of Automatic Control*, 42:674–690, 1997.
34. M. Turcotte, S. H. Muggleton, and M. J. E. Sternberg. The Effect of Relational Background Knowledge on Learning of Protein Three-Dimensional Fold Signatures. *Machine Learning Journal*, 43(1/2):81–95, 2001.