Machine Learning, 43, 53–80, 2001
© 2001 Kluwer Academic Publishers. Manufactured in The Netherlands.

Relational Instance-Based Learning with Lists and Terms

TAMÁS HORVÁTH tamas.horvath@gmd.de
German National Research Center for Information Technology, AiS.KD, Schloß Birlinghoven, D-53754 Sankt Augustin, Germany

STEFAN WROBEL wrobel@iws.cs.uni-magdeburg.de
Otto-von-Guericke-Universität Magdeburg, School of Computer Science, IWS, P.O.Box 4120, D-39106 Magdeburg, Germany

UTA BOHNEBECK bohnebec@informatik.uni-bremen.de
University of Bremen, Center for Computing Technologies, P.O.Box 330 440, D-28834 Bremen, Germany

Editor: Lorenza Saitta

Abstract. The similarity measures used in first-order IBL so far have been limited to the function-free case. In this paper we show that a lot of power can be gained by allowing lists and other terms in the input representation and designing similarity measures that work directly on these structures. We present an improved similarity measure for the first-order instance-based learner RIBL that employs the concept of *edit distances* to efficiently compute distances between lists and terms, discuss its computational and formal properties, and empirically demonstrate its additional power on a problem from the domain of biochemistry. The paper also includes a thorough reconstruction of RIBL's overall algorithm.

Keywords: inductive logic programming, relational instance-based learning

1. Introduction

For propositional learning problems instance-based learning (IBL) algorithms have always been a very popular and well studied choice (Aha, 1997; Aha, Kibler & Albert, 1991; Wettschereck & Dietterich, 1995). An instance-based algorithm does not compute a generalization of the examples at learning time, but simply stores the examples or variants thereof and then arrives at a classification based on comparing a new instance to the previously stored instances. Since generalization is delayed until classification time, such algorithms are also referred to as *lazy learning* algorithms (Aha, 1997; Aha, Kibler & Albert, 1991). Even the simplest instance-based learning algorithm, the k-nearest neighbor (kNN) classifier, is known to outperform other approaches for many application problems. More advanced variants address the problem of sensitivity to irrelevant features by learning feature weights (Wettschereck, Mohri & Aha, 1997) and the problem of overly large instance sets by storing selected examples or generalized prototypes (Aha, Kibler & Albert, 1991; Salzberg, 1991).

In first-order relational learning, as it is studied e.g. in the field of Inductive Logic Programming (ILP) (Muggleton & De Raedt, 1994; Wrobel, 1996) the use of instance-based learning techniques was first described in Emde and Wettschereck (1996), and implemented in the first version of the Relational Instance-Based Learning system RIBL. Even though this original version of RIBL was limited to function-free relational representations, it was applied very successfully to practical problems such as the classification of Diterpene molecules based on their nuclear magnetic resonance (NMR) spectra (Džeroski et al., 1996).

In this paper we present the most recent version of RIBL which has been extended to now handle relational representations with lists and other functional terms as well. To this end we define an improved similarity measure for RIBL that employs the concept of *edit distances* to efficiently compute distances between lists and terms. With experiments in a real-world domain (signal structure prediction in mRNA-molecules) we show that the new similarity measure allows significant additional power to be gained. The paper also includes a thorough reconstruction of the basic original RIBL algorithm and can thus serve as a precise reference to RIBL which has previously only been described in the conference paper (Emde & Wettschereck, 1996).

The paper is organized as follows: In the next section we briefly review the basics of IBL and propositional similarity measures. In Section 3, we then give a detailed reconstruction of the RIBL algorithm that supplements the original paper of (Emde & Wettschereck, 1996). Section 4 then presents the main technical contribution of this paper, the new similarity measure for constants, lists and terms. We discuss its computational properties and prove that the measure is a semi-metric. Section 5 is devoted to the empirical evaluation on the mRNA signal structure prediction problem first presented in Bohnebeck, Horváth and Wrobel (1998a). In Section 6 we discuss related work. Section 7 contains our conclusions and pointers to future work.

2. Instance-based learning and RIBL

In function learning from examples, a learner is given a set of examples E from an instance space $\mathcal{I}$ along with a class label or target value $c(e) \in Y$ for each $e \in E$. If Y contains only a few discrete values, one often speaks of a classification learning problem; if Y has only two values, the task is referred to as concept learning, and one of the target function values (e.g., $c(e) = 1$) is taken to mean "e is a member of the target concept", and the other value (e.g., $c(e) = 0$) is taken to mean "e is not a member of the target concept". If Y is a continuous set such as $\mathbb{R}$, the problem is also named a regression learning problem. From these examples, most inductive learners would induce a hypothesis h that is a generalization of the examples, e.g., a decision tree, and use it to classify new, unseen examples, i.e., the learner's prediction c' for a new instance $e_{new} \in \mathcal{I}$ would be

$$c'(e_{new}) := h(e_{new}).$$

An *instance-based* learner, on the other hand, in its simplest form just stores all of the examples E. It then bases its prediction on those k examples from E that are "closest" to e_{new} (where k is a user-given or system determined parameter). To this end, the instance-based learner possesses (or is given) a *distance measure* DIST

$$\textsc{Dist} : \mathcal{I} \times \mathcal{I} \rightarrow \mathbb{R}$$

that assigns a real number to each pair of instances such that "closer" pairs of instances are assigned smaller numbers. In the following, we will assume that DIST is bounded between 0 and 1, so we can equivalently speak of the *similarity measure* SIM defined as

$$\textsc{Sim}(x, y) := 1 - \textsc{Dist}(x, y).$$

The learner uses SIM (or DIST) to determine the *k nearest neighbors* $N(e_{new}) = \{e_{n_1}, \ldots, e_{n_k}\} \subseteq E$ of e_{new} (breaking ties arbitrarily), and lets these nearest neighbors "vote" to arrive at a prediction for e_{new}. Usually, each neighbor's vote is weighted according to its distance, so that closer neighbors have larger influence.

For a discrete prediction problem with class values $Y = \{c_1, \ldots, c_m\}$, we select the class which individually has received the highest weighted vote. More precisely, let $N_{c_j}(e_{new}) := \{e \in N(e_{new}) \mid c(e) = c_j\}$ denote the neighbors of e_{new} voting for class c_j. Then the voting procedure is:

$$h(e_{new}) := \text{argmax}_{c_j \in C} \sum_{e \in N_{c_j}(e_{new})} \textsc{Sim}(e_{new}, e).$$

For a continuous prediction problem, instead of selecting the value that received the highest vote, we predict the weighted average of the neighbors' class values:

$$h(e_{new}) := \frac{\sum_{e \in N(e_{new})} \textsc{Sim}(e_{new}, e) \cdot c(e)}{\sum_{e \in N(e_{new})} \textsc{Sim}(e_{new}, e)}.$$

For obvious reasons, this simplest instance-based learner is also called a *k-nearest neighbor classifier* (kNN). The value k is usually determined by the learner using cross validation on the training set, choosing the value of k that performs best.

Since an instance-based learner delays generalization until classification time, such algorithms are also referred to as *lazy learning* algorithms. Based on the kNN algorithm, various more sophisticated approaches have been studied to address some of the open issues with kNN. First, for large example sets, storing all the examples might require too much storage and too much time for finding the nearest neighbors. This problem can be addressed by storing only selected examples (Aha, Kibler & Albert, 1991) or using generalized prototypes or rectangles in instance space (Salzberg, 1991). Second, the problem of sensitivity to irrelevant features has been addressed by different weight learning schemes (Wettschereck, Mohri & Aha, 1997).

2.1. *Propositional similarity measures*

Since the similarity measure is the central element of an instance-based learner, let us have a closer look at its required properties. These are easier to express for distance measures.

Ideally, a distance measure should be a *metric*:

$$\textsc{Dist}(x, y) \geq 0 \tag{1}$$
$$\textsc{Dist}(x, y) = 0 \text{ iff } x = y \tag{2}$$
$$\textsc{Dist}(x, y) = \textsc{Dist}(y, x) \tag{3}$$
$$\textsc{Dist}(x, y) \leq \textsc{Dist}(x, z) + \textsc{Dist}(z, y) \tag{4}$$

for every $x, y, z \in \mathcal{I}$. A metric DIST is said to be *trivial* if for every $x, y \in \mathcal{I}$

$$\textsc{Dist}(x, y) = \begin{cases} 0 & \text{if } x = y \\ 1 & \text{otherwise.} \end{cases}$$

DIST is a *semi-metric* if Condition (4) (the *triangle inequality*) does not hold. Relaxing Condition (2) to $\textsc{Dist}(x, x) = 0$ for $\forall x \in \mathcal{I}$, we get the definition of *pseudo-metric*. A metric, semi-metric or pseudo-metric DIST is r*-bounded* ($r \in \mathbb{R}$) if r is an upper bound on the values of DIST. In this paper, all distance (and thus similarity) measures considered are 1-bounded.

Depending on how instances in $\mathcal{I}$ are represented, defining a distance measure (or similarity measure, respectively) that satisfies the above conditions is more or less difficult. For propositional representations including both numeric and discrete attributes, one usually defines distance measures on individual attributes depending on their type.

Let $\textsc{Type}(A)$ denote the type of an attribute A. For a numerical attribute A with value range $[a, b]$ ($\textsc{Type}(A) := \texttt{number}(a, b)$), we define the similarity of two attribute values r_1 and r_2 as

$$\textsc{Sim}_{att}(r_1, r_2, A) = 1 - \frac{|r_1 - r_2|}{b - a}.$$

For a discrete attribute A ($\textsc{Type}(A) := \texttt{discrete}$), we define the similarity of two attribute values d_1 and d_2 by

$$\textsc{Sim}_{att}(d_1, d_2, A) = \begin{cases} 1 & \text{if } d_1 = d_2 \\ 0 & \text{otherwise.} \end{cases}$$

From these, the overall similarity of two instances is computed as the average of the similarities of all attributes. More precisely, let $A_1, \ldots, A_n$ denote the attributes describing the instances, and let $i[j]$ denote the value of attribute A_j for a particular instance i. Then define the similarity of two instances $i_1, i_2 \in \mathcal{I}$ as

$$\textsc{Sim}(i_1, i_2) := \frac{\sum_{j=1,\ldots,n} \textsc{Sim}(i_1[j], i_2[j], A_j)}{n}.$$

We note that if desired, attribute weights may be assigned or learned (Wettschereck, Mohri & Aha, 1997); in this case, the weighted average is used in the above formula.

3. First-order instance-based learning in RIBL

RIBL, as first introduced in Emde and Wettschereck (1996), is a first-order instance-based learner that applies the basic principles of IBL as described above to an instance space consisting of first-order descriptions. More precisely, RIBL uses a k-nearest-neighbor algorithm that determines the value of k through cross-validation on the training set. It differs from the basic algorithm firstly through its first-order learning task and similarity measure, and secondly through its use of a special weight learning scheme for predicates and arguments. Due to space reasons, we do not describe this weight learning approach here, and refer the reader to Emde and Wettschereck (1996) for details. Instead, in this section, we give a detailed definition of the learning task addressed by RIBL and then provide a thorough reconstruction of the first-order similarity measure that supplements the short description in Emde and Wettschereck (1996).

3.1. Learning task

RIBL turns the basic instance-based learning problem into a first-order version by allowing each instance to be described not only by numerical and discrete attributes, but also by attributes of type object ($\text{TYPE}(A) = \texttt{object}$) containing object identifiers that point to further information in a separately specified body of *background knowledge* $\mathcal{B}$, a database consisting of first-order ground atoms (first-order facts). Let us first illustrate this with an example.

Assume we want to send a culinary present to a friend, and want to classify a number of prepackaged present sets from different vendors according to whether they are *suitable* for our friend or not. Based on a few manually classified examples, we want further offers to be classified automatically by RIBL. Assume that about each set, we know its price and delivery mode (personal, mail, or pick-up). Each instance will be represented by three attributes: its identifier, price, and delivery mode. In addition, we know the contents of each package which for simplicity we assume consist of different red wines and cheeses. We also know something about different vineyards. Thus, our first package might be represented as follows:

$$e\ :\ (set1, 125, personal) \text{ with } c(e) = suitable,$$

whereas a new package to be classified might be represented by

$$e_{new}\ :\ (set25, 195, mail) \text{ without classification, i.e., } c(e_{new}) = ?.$$

In these instances the first argument (set1 and set25, respectively) is of type `object` and refers to further information in the background knowledge $\mathcal{B}$, such as:

% set ID, cheese type, weight, and origin of cheese
cheese(*set*1, *camembert*, 150, *france*)
cheese(*set*25, *roquefort*, 200, *france*)
cheese(*set*25, *ricotta*, 100, *italy*)

% set ID, vineyard ID, vintage year, and bottle size
wine(*set*1, *mouton*, 1988, 0.75)
wine(*set*1, *gallo*, 1995, 0.5)
wine(*set*25, *mouton*, 1995, 0.75)

% vineyard ID, popularity, size, country
vineyard(*gallo*, *famous*, *large*, *usa*)
vineyard(*mouton*, *famous*, *small*, *france*)

Note that $\mathcal{B}$ contains both general background knowledge and knowledge specific to individual instances. This particular example uses the function-free representation of Emde and Wettschereck (1996); below, we also allow arguments of type `constant` (with custom defined distances), `list` (which may contain lists), and arguments of type `term` (which may contain arbitrary first-order terms). Thus, the exact task considered in this paper is defined as follows.

Given:

- a set of examples E, each described by a set of attributes $\{A_1, \ldots, A_n\}$ of type `number`, `discrete`, `object`, `constant`, `list`, or `term`,
- a target value[1] $c(e)$ for every $e \in E$,
- background knowledge $\mathcal{B}$ consisting of first-order atoms with arguments of type `number`, `discrete`, `object`, `list`, or `term`, and
- a set of unseen instances E_{new} each described by the same attributes as E.

Find: A prediction $c'(e)$ of the target value for each $e \in E_{new}$.

3.2. Similarity Computation in RIBL

To compute the similarity of two first-order instances, naturally we want to take into account not only the attributes of the instances, but also the available background knowledge. To this end, RIBL uses a recursive descent strategy similar to the one employed in Bisson (1992b). We first illustrate the idea using our wine and cheese running example (see also figure 1). So assume we want to compare e and e_{new}. For the arguments expressing price and delivery mode, we simply rely on the standard distances for numerical and discrete attributes. To compare the object identifiers *set*1 and *set*25, however, we first collect all we know about these two objects from the background knowledge. In our example we obtain the two sets $\mathcal{L}_{set1}$ and $\mathcal{L}_{set25}$ of logical atoms

$$\begin{aligned}\mathcal{L}_{set1} = \{&\text{cheese}(\mathit{set}1, \mathit{camembert}, 150, \mathit{france}),\\ &\text{wine}(\mathit{set}1, \mathit{mouton}, 1988, 0.75),\\ &\text{wine}(\mathit{set}1, \mathit{gallo}, 1995, 0.5)\},\end{aligned}$$

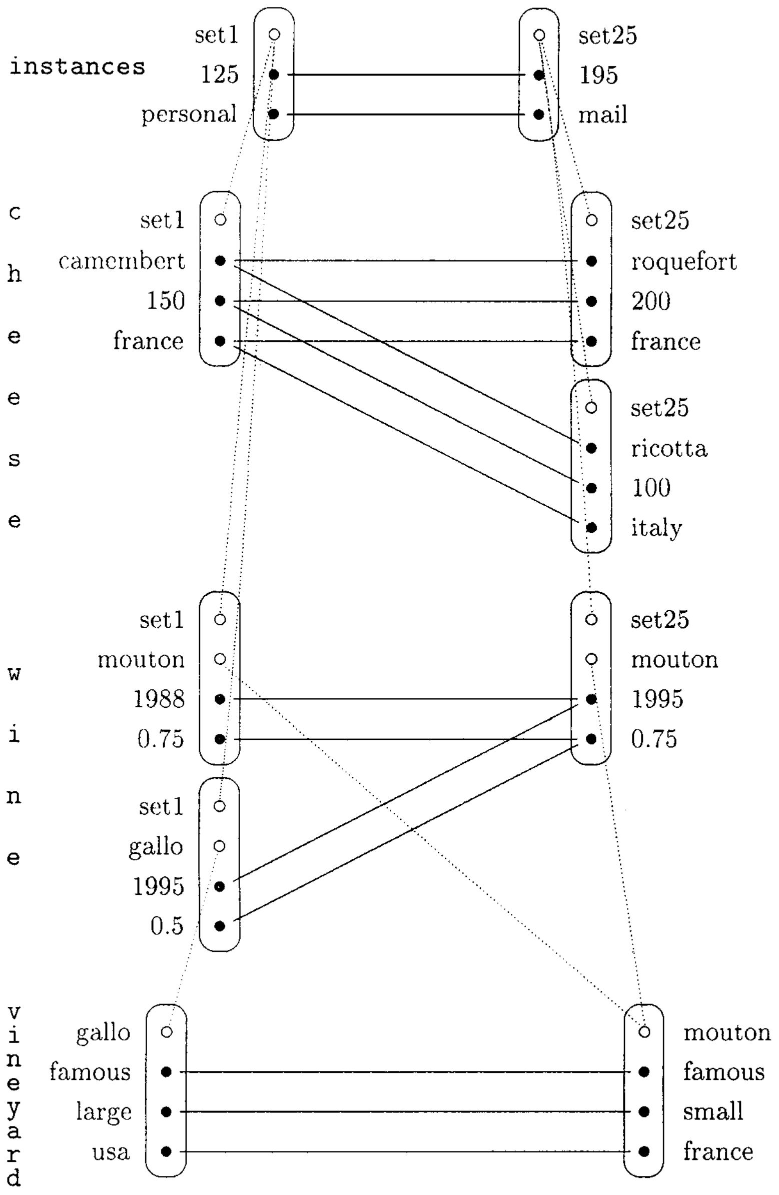


Figure 1. The recursive scheme for computing the similarity between e and e_{new}.

$$\begin{aligned}\mathcal{L}_{set25} = \{&\text{cheese}(set25, roquefort, 200, france)\\ &\text{cheese}(set25, ricotta, 100, italy)\\ &\text{wine}(set25, mouton, 1995, 0.75)\},\end{aligned}$$

respectively.

We now have to compute the similarity between these two sets of atoms. We do this separately for each predicate that is used, and then average across all of them. So taking the atoms using “wine” as an example, we get the subsets

$$\mathcal{L}'_{set1} = \{\text{wine}(set1, mouton, 1988, 0.75), \text{wine}(set1, gallo, 1995, 0.5)\},$$
$$\mathcal{L}'_{set25} = \{\text{wine}(set25, mouton, 1995, 0.75)\},$$

respectively. For each atom in the smaller set, which in this case is the set $\mathcal{L}'_{set25}$ for e_{new}, we determine which of the atoms in the other set is most similar. Thus in our example we compute the similarity between

wine(*set*25,*mouton*,1995,0.75) and wine(*set*1,*mouton*,1988,0.75),

and between

wine(*set*25,*mouton*,1995,0.75) and wine(*set*1,*gallo*,1995,0.5).

To do this, we apply the same principle recursively: We treat the two atoms as instances to be compared (however, we ignore the first argument to avoid infinite recursion). Thus in our example, we use the basic similarity measures on the third and fourth argument and collect further background knowledge about the objects *mouton* and *gallo*, i.e., for the second pair of atoms we need to compare the sets of literals

$\{\text{vineyard}(gallo, famous, large, usa)\}$ and
$\{\text{vineyard}(mouton, famous, small, france)\}$.

Once we know the most similar partner for each literal of the smaller set, we sum up all these similarities and normalize with the size of the *larger* set. This is done to ensure that with an increasing number of extra literals in one of the sets, similarity becomes smaller (if one of the sets is empty, similarity is 0 for this pair of atom sets). In our example the most similar atom for wine (*set*25,*mouton*,1995,0.75) is wine (*set*1,*mouton*,1988,0.75), the similarity of which is divided by two since the larger set contained two atoms. A similar computation is made for the cheese literals, arriving at the overall similarity for *set*1 and *set*25 by adding the cheese and wine literal set similarities, and dividing by two.

In our example, we have used all of the available background knowledge; for larger problems, a depth bound is used to ensure limited effort for the recursive computation.

We precisely define the similarity computation as follows. Let $\mathcal{B}(e, 1)$ denote the set of background facts containing at least one object from the arguments of an instance e, and let $\mathcal{B}(e, d+1) \subseteq \mathcal{B} \setminus \bigcup_{i=1}^{d} \mathcal{B}(e, i)$ denote the set of facts containing at least one object occurring in a fact from $\mathcal{B}(e, d)$. Note that the above language includes propositional representations as special cases. For instance, in our example, $\mathcal{B}(e_{new}, 1)$ is the set $\mathcal{L}_{set25}$.

The similarity function used by RIBL to compare two instances e_1 and e_2 with attributes $A_1, \ldots, A_n$ with respect to background knowledge $\mathcal{B}$ is defined as

$$\textsc{Sim}(\mathcal{B}, e_1, e_2) := \frac{\sum_{j=1,\ldots,n} \textsc{Similarity}(\mathcal{B}, e_1, e_2, 0, e_1, e_2, j)}{n}. \tag{5}$$

The algorithmic definition of the term SIMILARITY in (5) is given in Algorithm 1. In the algorithm we generally use $e[i]$ to denote the i-th argument value of an atom e, or the i-th element of a list e. Furthermore, by a q-atom we mean an atom with predicate symbol q. The algorithm computes, in the context of comparing e_1 and e_2, the normalized similarity between the ith arguments of the atoms l_1, and l_2, where l_1, and l_2 are both of depth d with respect to e_1 and e_2. (For the sake of consistency, we define the depth of an instance to be 0, and also refer to instances as atoms.) We use the similarity measure of Section 2 for arguments of type `number`, and `discrete`, respectively (see Lines 3, and 5 in Algorithm 1). In order to compare objects o_1, and o_2 (i.e., arguments of type `object`), the algorithm first checks whether d has already reached the value of the depth parameter. If yes, it outputs 1, or 0 depending on the truth of $o_1 = o_2$ (Line 8). Otherwise, we first collect from $\mathcal{B}$ the set $\mathcal{L}_i$ of literals with depth $d+1$ containing o_i (Line 12) for $i = 1, 2$. The similarity between o_1, and o_2 is defined by the similarity between the sets $\mathcal{L}_1$, and $\mathcal{L}_2$. The comparison of two sets of facts is based on a similarity measure between two facts. In RIBL the similarity between two facts is reduced again to the similarity between arguments thus making the algorithm recursive. The formal definition of similarity between facts is given by $\textsc{Sim}_{\text{lit}}$ in Line 18.

Now we can turn back to the problem of similarity between the sets $\mathcal{L}_1$, and $\mathcal{L}_2$ induced by o_1, and o_2, respectively. Let $\mathcal{P}_i$ denote the set of pairs (q', k) for which there is a q'-atom in $\mathcal{L}_i$ such that its kth argument is o_i for $i = 1, 2$. For each argument position (q', k) from $\mathcal{P}_1 \cup \mathcal{P}_2$ we collect the q'-atoms from $\mathcal{L}_1$ (resp. $\mathcal{L}_2$) with kth argument o_1 (resp. o_2). From these two sets let $\mathcal{L}'_{\text{max}}$ denote the set that has greater or equal cardinality, and $\mathcal{L}'_{\text{min}}$ the other one (Lines 16–17). For each atom from $\mathcal{L}'_{\text{min}}$ we choose its most similar atom from $\mathcal{L}'_{\text{max}}$, and take the sum of the similarities of these pairs of atoms divided by the cardinality of $\mathcal{L}_{\text{max}}$ (fraction in Line 18). Finally, the similarity between o_1, and o_2 is computed as the sum of the above similarities for every $(q', k) \in \mathcal{P}_1 \cup \mathcal{P}_2$ divided by the cardinality of $\mathcal{P}_1 \cup \mathcal{P}_2$ (Line 20).

4. A similarity measure for lists and terms

In order to directly computer similarity of instances with non-flat components (lists, terms), we have chosen to rely on the idea of *edit distances* between objects. Intuitively, with an edit distance, we are given a set of edit operations (insert, delete, change) with an associated cost function δ that tells us how expensive it is to delete or insert an element of our alphabet Σ, or how expensive it is to change one element into another. The edit distance is given by the smallest cost sequence of such operations that turns the first object into the second, and there are efficient algorithms for computing it (see below).

```
Input: $l_1, l_2$ are $q$-atoms of depth $d$, $1 \le i \le \rho(q)$.
Output: Normalized similarity between the $i$th arguments of $l_1, l_2$,
    respectively.
 1: let $o_1 = l_1[i]$, $o_2 = l_2[i]$
 2: if $\texttt{Type}(q[i]) = \texttt{number(a,b)}$ then
 3:     return $1 - \frac{|o_1 - o_2|}{b - a}$
 4: else if $\texttt{Type}(q[i]) = \texttt{discrete}$ then
 5:     return 1 if $o_1 = o_2$, and 0 otherwise
 6: else if $\texttt{Type}(q[i]) = \texttt{object}$ then
 7:     if $d = \textsc{Depth}$ then
 8:         return 1 if $o_1 = o_2$, and 0 otherwise
 9:     else if $o_1 = o_2$ then
10:         return 1
11:     else
12:         let $\mathcal{L}_j = \{l \in \mathcal{B}(e_j, d+1) : o_j \text{occurs in } l\}$
13:         for $j = 1, 2$ let

              $\mathcal{P}_j = \{(q', k) : q'$ is a background predicate, $1 \le k \le \rho(q')$,
                   there exists a $q'$-atom $l \in \mathcal{L}_j$ such that $l[k] = o_j\}$

14:           $S = 0$
15:           for all $(q', k) \in \mathcal{P}_1 \cup \mathcal{P}_2$ do
16:               let $\mathcal{L}'_j = \{l \in \mathcal{L}_j : l$ is a $q'$-atom, and $l[k] = o_j\}$ for $j = 1, 2$
17:               let $\mathcal{L}'_{\min}$ be the smaller set from $\mathcal{L}'_1, \mathcal{L}'_2$, and let $\mathcal{L}'_{\max}$ denote
                  the other one
18:               $S = S + \frac{\sum_{l_x \in \mathcal{L}'_{\min}} \max_{l_y \in \mathcal{L}'_{\max}} \textsc{Sim}_{\text{lit}}(\mathcal{B}, e_1, e_2, d+1, l_x, l_y)}{|\mathcal{L}'_{\max}|}$
              where

                  $\textsc{Sim}_{\text{lit}}(\mathcal{B}, e_1, e_2, d+1, l_x, l_y) =$
                      $\frac{1}{|I|} \sum_{i' \in I} \textsc{Similarity}(\mathcal{B}, e_1, e_2, d+1, l_x, l_y, i')$
              with $I = \{u : 1 \le u \le \rho(q'), l_x[u] \ne o_1 \vee l_x[u] \ne o_2\}$
19:           end for
20:           return $\frac{S}{|\mathcal{P}_1 \cup \mathcal{P}_2|}$
21:     end if
22: end if
```

Algorithm 1: $\textsc{Similarity}(\mathcal{B}, e_1, e_2, d, l_1, l_2, i)$

In the following, we will precisely specify the edit distance measure used in the current version of RIBL (henceforth referred to as RIBL 2.0). Even though both constants and lists could be considered as terms, we distinguish them from each other for the sake of efficiency (see Section 4.4).

4.1. *Constants*

An argument type `constant` is given by a set of constant symbols Σ_{constant} and a 1-bounded semi-metric δ on Σ_{constant}. The *distance* between $a, b \in \Sigma_{\text{constant}}$ is then defined directly by δ, i.e.,

$$\text{DIST}_{\text{constant}}(a, b) = \delta(a, b).$$

We note that the argument type `discrete` can be considered as a special case of this type.

4.2. *Lists*

Let $\mathcal{L}(\Sigma_{\text{list}})$ be the set of finite lists over the alphabet Σ_{list}, and let $\Sigma_{\text{list},\perp} = \Sigma_{\text{list}} \cup \{\perp\}$, where $\perp$ denotes the *empty list*. $|l|$ denotes the *length* of the list $l \in \mathcal{L}(\Sigma_{\text{list}})$. Consider the following operations on the elements of $\mathcal{L}(\Sigma_{\text{list}})$:

– *insert*(l, i, a): *inserts* the symbol $a \in \Sigma_{\text{list}}$ before the i-th symbol in l,
– *delete*(l, i): *deletes* the i-th symbol in l,
– *change*(l, i, a): *changes* the i-th symbol in l to a *new* symbol a

for every $a \in \Sigma_{\text{list}}$, $l \in \mathcal{L}(\Sigma_{\text{list}})$ and $i = 1, \ldots, |l|$. Thus, an operation can be considered as an application of a *rewriting rule* $a \to b$ for $a, b \in \Sigma_{\text{list},\perp}$, i.e., insert, delete and change correspond to the application of a rewriting rule $\perp \to a$, $a \to \perp$ and $a \to b$, respectively $(a, b \neq \perp)$.

Let $l_1, l_2 \in \mathcal{L}(\Sigma_{\text{list}})$. We say that l_2 can be *directly derived* from l_1 (denoted by $l_1 \Rightarrow l_2$) if there exists an operation that produces l_2 from l_1. The *derivation* (denoted by $l_1 \Rightarrow^*_{\sigma_1\cdots\sigma_n} l_2$) is the *reflexive, transitive closure* of $\Rightarrow$. The index $\sigma_1 \cdots \sigma_n$ denotes the sequence of the applied rewriting rules. A `list` type is given by a 1-bounded metric space $(\Sigma_{\text{list},\perp}, \delta)$. Here, $\delta(a \to b)$ is considered as the *cost* of the rewriting rule $a \to b$ for every $a, b \in \Sigma_{\text{list},\perp}$. Since δ is a metric, it can be given by a symmetric *cost matrix* with zero elements in its diagonal. The *edit* or *Levenshtein distance*[2] between $l_1, l_2 \in \mathcal{L}(\Sigma_{\text{list}})$ is defined by

$$\text{EDITDIST}_{\text{list}}(l_1, l_2) = \min\left\{ \sum_{i=1}^{n} \delta(\sigma_i) \ : \ \exists \sigma_1, \ldots, \sigma_n \text{ such that } l_1 \Rightarrow^*_{\sigma_1 \ldots \sigma_n} l_2 \right\}.$$

Since δ is a distance metric on $\Sigma_{\text{list},\perp}$, $\text{EDITDIST}_{\text{list}}$ is a distance metric on $\mathcal{L}(\Sigma_{\text{list}})$. The next theorem is a special case of Theorems 2 and 3 in Wagner and Fischer (1974):

Theorem 1. *Let* $l_1 = [a_1, \ldots, a_n], l_2 = [b_1, \ldots, b_m] \in \mathcal{L}(\Sigma_{\text{list}})$, *and* $(\Sigma_{\text{list},\perp}, \delta)$ *be a metric space. Let* $D_{i,j}$ *denote the edit distance between* $[a_1, \ldots, a_i]$, *and* $[b_1, \ldots, b_j]$

Table 1. The distance matrix $D_{i,j}$.

	$\perp$	c	u	u	a	g	c	u	c
$\perp$	0	1	2	3	4	5	6	7	8
c	1	0	1	2	3	4	5	6	7
c	2	1	1	2	3	4	4	5	6
u	3	2	1	1	2	3	4	4	5
a	4	3	2	2	1	2	3	4	$\underline{5}$

for $i = 0, \ldots, n$, *and* $j = 0, \ldots, m$. *Then*

$$D_{0,0} = 0,\ D_{0,j} = \sum_{k=1}^{j} \delta(\perp \to b_k),\ D_{i,0} = \sum_{k=1}^{i} \delta(a_k \to \perp),\ \textit{and}$$
$$D_{i,j} = \min(D_{i-1,j-1} + \delta(a_i \to b_j),\ D_{i-1,j} + \delta(a_i \to \perp), D_{i,j-1} + \delta(\perp \to b_j))$$

for every $i, j > 0$.

By Theorem 1, $\textsc{EditDist}_{\text{list}}(l_1, l_2)$ can be computed in *time* and *space* $O(mn)$.

Example 2. Let $\Sigma_{\text{list}} = \{a, c, g, u\}$, and δ be the trivial metric on $\Sigma_{\text{list},\perp}$. Let $l_1 = [c, c, u, a]$, and $l_2 = [c, u, u, a, g, c, u, c]$. The value of $D_{i,j}$ for $i = 0, \ldots, 4$, and $j = 0, \ldots, 8$ is given in Table 1. The edit distance is $\textsc{EditDist}_{\text{list}}(l_1, l_2) = D_{4,8} = 5$.

An improved algorithm for computing the edit distance between lists and the next theorem can be found in Ukkonen (1985).

Theorem 3. *The edit distance* $d = \textsc{EditDist}_{\text{list}}(l_1, l_2)$ *between lists* l_1 *and* l_2 *can be computed in time and space*

$$O(d \cdot \min(|l_1|, |l_2|)) \textit{ and } O(\min(d, |l_1|, |l_2|)),$$

respectively.

We define distance between lists in RIBL as:

$$\textsc{Dist}_{\text{list}}(l_1, l_2) = \frac{\textsc{EditDist}_{\text{list}}(l_1, l_2)}{\max(|l_1|, |l_2|)}. \tag{6}$$

Proposition 4. *If* $(\Sigma_{\text{list},\perp}, \delta)$ *is a* 1*-bounded metric space then* $\textsc{Dist}_{\text{list}}$ *is a* 1*-bounded semi-metric on* $\mathcal{L}(\Sigma_{\text{list}})$.

Proof: Since $\text{EDITDIST}_{\text{list}}$ is a metric on $\mathcal{L}(\Sigma_{\text{list}})$, $\text{DIST}_{\text{list}}$ satisfies Conditions (1)–(3) defined in Section 2.1. We show that (6) violates the triangle inequality. Consider the alphabet $\Sigma_{\text{list}} = \{a, b\}$ with the trivial metric δ. Then

$$\begin{aligned}\text{DIST}_{\text{list}}([a,b],[b,a]) &= 2/2\\ &> \text{DIST}_{\text{list}}([a,b],[a,b,a])\\ &\quad + \text{DIST}_{\text{list}}([a,b,a],[b,a])\\ &= 1/3 + 1/3.\end{aligned}$$

As δ is a 1-bounded metric on $\Sigma_{\text{list},\perp}$,

$$\text{EDITDIST}_{\text{list}}(l_1, l_2) \leq \max(|l_1|, |l_2|)$$

for every $l_1, l_2 \in \mathcal{L}(\Sigma_{\text{list}})$. Hence, $\text{DIST}_{\text{list}}$ is also 1-bounded. □

4.3. Terms

In RIBL 2.0, terms are considered as *labeled ordered trees* and hence, the distance between terms is computed as the *edit distance* between labeled ordered trees.

Let Σ_{term} be an alphabet. We note that Σ_{term} is not a *ranked* alphabet, i.e., no arity function is assigned to Σ_{term}. The set of all *finite* terms with nodes labeled by the elements of Σ_{term} including the *empty* term ($\perp$) is denoted by $\mathcal{T}(\Sigma_{\text{term}})$.

For a given term $t \in \mathcal{T}(\Sigma_{\text{term}})$, $|t|$ denotes the number of nodes of t, and $t[i]$ ($1 \leq i \leq |t|$) denotes the ith node of t according to the preorder traversal of t. Let t_1, and t_2 be terms from $\mathcal{T}(\Sigma_{\text{term}})$. A *mapping* from t_1 to t_2, denoted by $M_{t_1 \to t_2}$, is a subset of $\{1, \ldots, |t_1|\} \times \{1, \ldots, |t_2|\}$ such that for every $(i_1, j_1), (i_2, j_2) \in M$ it holds that

(i) $i_1 = i_2$ iff $j_1 = j_2$,
(ii) $i_1 < i_2$ iff $j_1 < j_2$,
(iii) $t_1[i_1]$ is an ancestor (resp. descendant) of $t_2[i_2]$ iff $t_1[j_1]$ is an ancestor (resp. descendant) of $t_2[j_2]$.

A mapping $M_{t_1 \to t_2}$ can be considered as a one-to-one correspondence between subsets of nodes of t_1 and t_2, respectively, such that by deleting the 'untouched' nodes from t_1 and t_2, respectively, the remaining terms are identical up to relabeling. For a detailed discussion of mapping the reader is referred to Tai (1979). If there is no confusion we define a mapping by using the nodes' labels instead of their positions according to the preorder walk. Furthermore, if t_1 and t_2 follow from the context we write M instead of $M_{t_1 \to t_2}$.

Example 5. Consider the terms

$$t_1 = a(b, c(d, e(f, g), h)) \quad \text{and} \quad t_2 = a'(c(d, e(f, g), h), b').$$

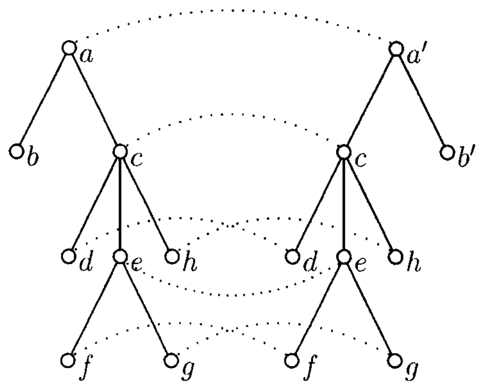


Figure 2. Mapping between two terms.

Then $M_1 = \{(a, a'), (b, c)\}$ is a mapping from t_1 to t_2, as the corresponding positions $\{(1, 1), (2, 2)\}$ according to preorder traversal satisfy conditions (i)–(iii). Note that the trees $a(b)$ and $a'(c)$ obtained from t_1 by deleting the nodes c, d, e, f, g, h and from t_2 by deleting b', d, e, f, g, h, respectively, are identical up to relabeling.

$M_2 = \{(a, a'), (c, c), (d, d), (e, e), (h, h), (f, f), (g, g)\}$ is also a mapping between t_1, and t_2 (see figure 2).

A mapping M between terms t_1, and t_2 defines a transformation from t_1 into t_2 by the following sequence of *edit* operations:

– *delete* each untouched node from t_1 (i.e., each node x of t_1 such that there is no pair $(x, y) \in M$ for any y), then
– *insert* each untouched node of t_2 with its corresponding label (i.e., each node y of t_2 such that there is no pair $(x, y) \in M$ for any x), and then
– for every $(x, y) \in M$ such that x, y have different labels, *replace* the label of x by the label of y.

The above edit operations on terms can also be considered as applications of *term rewriting rules* $a \rightarrow b$ ($a, b \in \Sigma_{\text{term},\perp}$ and $a \rightarrow b \neq \perp \rightarrow \perp$). Let δ be a 1-bounded metric on $\Sigma_{\text{term},\perp}$. As for lists, δ is considered as a *cost matrix* for the term rewriting rules. The *cost* of a mapping M between t_1, t_2 is the sum of the costs of the applied term rewriting rules transforming t_1 into t_2. The *edit distance* between t_1, t_2, denoted by $\text{EDITDIST}_{\text{term}}(t_1, t_2)$, is the cost of the mapping with the smallest cost with respect to δ. Since $(\Sigma_{\text{term},\perp}, \delta)$ is a metric space, $(\mathcal{T}(\Sigma_{\text{term}}), \text{EDITDIST}_{\text{term}})$ is also a metric space.

The algorithm for computing the edit distance between terms and the following theorem can be found in Zhang and Shasha (1989) (we denote by *depth*(t), and *leaves*(t) the depth, and the number of leaves of the term t, respectively):

Theorem 6. *The edit distance between terms t_1 and t_2 can be computed in time*

$$O(|t_1| \times |t_2| \times \min(depth(t_1), leaves(t_1)) \times \min(depth(t_2), leaves(t_2)))$$

and space $O(|t_1| \times |t_2|)$.

In contrast to lists, in the next example we show that $\max(|t_1|, |t_2|)$ is *not* an *upper bound* on $\text{EDITDIST}_{\text{term}}$.

Example 7. Let δ be the trivial metric on $\{\perp, a, b, c\}$, and consider the terms $t_1 = a(a \cdots (a)) \cdots)$ (i.e., a *chain* of a's) and $t_2 = b(c, \ldots, c)$ with $|t_1| = |t_2| = n$. The edit distance between the two terms is $2n - 2$.

Although $|t_1| + |t_2|$ is an upper bound on $\text{EDITDIST}_{\text{term}}(t_1, t_2)$ for every 1-bounded metric space $(\Sigma_{\text{term},\perp}, \delta)$, and $t_1, t_2 \in \mathcal{T}(\Sigma_{\text{term}})$, we normalize with another function measuring the *structural difference* of t_1, and t_2 and giving a tighter upper bound.

Let $M_{t_1 \to t_2}$ be a mapping. By the definition of mapping, M is a set of ordered pairs. We denote by $\mathcal{U}_1(M)$ (resp. $\mathcal{U}_2(M)$) the set of nodes of t_1 (resp. t_2) *untouched* by M. The *size* of a mapping is defined by

$$size(M) = |M| + |\mathcal{U}_1(M)| + |\mathcal{U}_2(M)|. \tag{7}$$

The *optimal mapping* $M^{\text{OPT}}_{t_1 \to t_2}$ is a mapping with the smallest size.

In RIBL distance between terms is defined by

$$\text{DIST}_{\text{term}}(t_1, t_2) = \frac{\text{EDITDIST}_{\text{term}}(t_1, t_2)}{size\big(M^{\text{OPT}}_{t_1 \to t_2}\big)}. \tag{8}$$

Example 8. Let $t_1 = a(b, c(d))$, and $t_2 = a(b(d))$ be terms.

$$\text{EDITDIST}_{\text{term}}(t_1, t_2) = 2$$

using the trivial metric. The optimal mapping is

$$M^{\text{OPT}}_{t_1 \to t_2} = \{(a, a), (c, b), (d, d)\}.$$

Since $\mathcal{U}_1\big(M^{\text{OPT}}_{t_1 \to t_2}\big) = \{b\}$, and $\mathcal{U}_2\big(M^{\text{OPT}}_{t_1 \to t_2}\big) = \emptyset$,

$$size\big(M^{\text{OPT}}_{t_1 \to t_2}\big) = 3 + 1 + 0$$

by (7). Applying (8) we get

$$\textsc{Dist}_{\text{term}}(t_1, t_2) = 2/4.$$

In RIBL, $size\big(M^{\text{OPT}}_{t_1\to t_2}\big)$ is computed as follows: let t_1/a (resp. t_2/b) denote the terms obtained from t_1 (resp. t_2) by replacing the label of each node with a (resp. b). Then

$$size\big(M^{\text{OPT}}_{t_1\to t_2}\big) = \textsc{EditDist}_{\text{term}}(t_1/a, t_2/b)$$

using the trivial metric on $\{\bot, a, b\}$.

Proposition 9. *If* $(\Sigma_{\text{term},\bot}, \delta)$ *is a* 1*-bounded metric space then* $\textsc{Dist}_{\text{term}}$ *is a* 1*-bounded semi-metric on* $\mathcal{T}(\Sigma_{\text{term}})$.

Proof: Since $\textsc{Dist}_{\text{term}}$ is a metric on $\mathcal{T}(\Sigma_{\text{term}})$, and the size of a mapping between terms is symmetric, (8) satisfy Conditions (1)–(3). However, it does not satisfy the triangle inequality, e.g. let $(\{\bot, a, b\}, \delta)$ be a trivial metric space, and consider the terms $a(b)$, and $b(a)$. Then

$$\begin{aligned}\textsc{Dist}_{\text{term}}(a(b), b(a)) &= 2/2\\ &> \textsc{Dist}_{\text{term}}(a(b), a(b(a)))\\ &+ \textsc{Dist}_{\text{term}}(a(b(a)), b(a))\\ &= 1/3 + 1/3.\end{aligned}$$

It remains to be proven that $\textsc{Dist}_{\text{term}}$ is 1-bounded. Let t_1, t_2 be terms, and $M_{t_1\to t_2}$ be any mapping. Since δ is 1-bounded,

$$size(M_{t_1\to t_2}) \geq \textsc{EditDist}_{\text{term}}(t_1, t_2). \tag{9}$$

As M is any arbitrary mapping from t_1 to t_2, (9) holds for $M^{\text{OPT}}_{t_1\to t_2}$ as well. □

4.4. Lists as terms

The standard term representation of lists is given by the function symbol $./2$, and the empty list $\bot$ (or [] in the standard Prolog notation). However, in this representation, $\textsc{Dist}_{\text{list}}$ and $\textsc{Dist}_{\text{term}}$ may differ on the same lists, e.g.

$$\begin{aligned}2/2 &= \textsc{Dist}_{\text{list}}([a, b], [b, a])\\ &\neq \textsc{Dist}_{\text{term}}(.(a, .(b, \bot)), .(b, .(a, \bot)))\\ &= 2/5\end{aligned}$$

using the trivial metric on $\{\bot, a, b\}$.

As an alternative, let $l = [a_1, \ldots, a_n]$ be a list in $\mathcal{L}(\Sigma_{\text{list}})$. The *chain term* representation of l is $a_1(a_2(\cdots(a_n)\cdots)$. Let l^c_1, and l^c_2 denote the chain representations of $l_1 = [a_1, \ldots, a_n]$, and $l_2 = [b_1, \ldots, b_m]$, respectively. Since $\text{EDITDIST}_{\text{list}}(l_1, l_2) = \text{EDITDIST}_{\text{term}}(l_1^c, l_2^c)$, and $\max(|l_1|, |l_2|) = size(M^{\text{OPT}}_{l_1^c \to l_2^c})$,

$$\text{DIST}_{\text{list}}(l_1, l_2) = \text{DIST}_{\text{term}}(l_1^c, l_2^c)$$

for every metric used in both representations.

Applying Theorem 6 to l_1^c, and l_2^c, the edit distance of lists in the chain term representation can be computed in $O(mn)$. The reason why we handle lists and terms separately is that the time and space complexity for computing edit distance between lists given in Ukkonen (1985) is always as good as the complexity of computing their edit distance in the chain term representation, and the norm function of lists can be computed in linear time.

4.5. Defining δ for the alphabets

In Sections 4.2 and 4.3, we have treated list and term distances as basic (non-recursive) distance functions which rely on metrics δ for comparing individual symbols from the alphabet. Clearly, one possibility is to use user-defined metrics which allows the encoding of additional domain knowledge of the user. Another attractive possibility is to use RIBL's basic idea again, and compute the editing cost of two symbols by using RIBL's similarity measure.

To do this, the elements of the corresponding alphabet must be considered as objects that are further described in the background knowledge. Each object is considered as an instance (without a target value). In contrast to the original IBL prediction problem, we are interested only in the similarities between such instances. These similarity values are then used for determining the editing costs between the corresponding symbols.

The editing costs δ on an alphabet Σ can be computed automatically by the following two steps:

1. we compute a *semi-metric* δ' on Σ by recursively recalling RIBL's similarity measure, i.e.,

$$\delta'(a, b) := 1 - \text{SIMILARITY}(\mathcal{B}, (a), (b), 0, (a), (b), 1)$$

(we use notation (x) for denoting the instance corresponding to x),

2. and then we transform it into a *metric* δ by

$$\delta(a, b) = \min(1, \delta'(a, b) + C) \tag{10}$$

for every $a, b \in \Sigma_{\text{term},\perp}$, where C is the maximum of

$$\frac{\delta'(x, y) - \delta'(x, z) - \delta'(z, y)}{2}$$

for every $x, y, z \in \Sigma$ violating the triangle inequality.

5. Empirical evaluation

In this section we discuss our experiments with the revised version of RIBL (RIBL 2.0) in a real-world domain, the *mRNA signal structure prediction* problem. The experiments are designed to show that the addition of lists and terms enables the easy construction of background knowledge representing complex objects, and that this representation leads to excellent results on a significant real-world problem, greatly improving on simpler, RIBL 1.0-style background knowledge. In fact, the mRNA application can also be considered a success in terms of the application domain (Bohnebeck et al., 1998b).

5.1. mRNA signal structures

Protein synthesis involves a *translation* step of copying specific regions (the genes) of DNA into a complementary strand of RNA. RNA molecules that *mediate* between the spatially distinct localizations of information storage (i.e., the DNA) and protein synthesis are called *messenger* RNA (mRNA). Beside the genetic information on the amino acid sequence of proteins, mRNA contains two regions, called *untranslated regions*, that do not code for protein. These untranslated regions contain continuous parts, called *signal structures*, that control gene expression at the posttranscriptional level. Depending on their biological functions (i.e., regulatory roles during protein synthesis), signal structures are grouped into different signal structure *classes*.

In general, RNA is composed of a linear sequence of the four bases *adenine* (A), *cytosine* (C), *guanine* (G), and *uracil* (U). This sequence is its *primary structure*. In contrast to DNA, RNA is a *single-stranded*, and relatively short molecule. The linear sequence of an RNA folds back on itself thus giving the RNA a *three-dimensional* conformation. The three-dimensional conformation contains local regions of short complementary base-pairing, i.e., the two stable Watson-Crick base pairs A-U, G-C, and the weak G-U. The *secondary structure* of an RNA is defined as the collection of the base pairs occurring in its three-dimensional structure (see Figure 3). Such a secondary structure can be viewed as a *tree* (Shapiro & Zhang, 1990) with nodes, also called *structure elements*, corresponding to continuous regions that belong to basic *patterns*.[3] This observation holds for signal structures as well (i.e., the secondary structure of a signal structure can be considered as a tree), as their primary structures are (continuous) subsequences of the RNA's primary structure (see also the right upper part of Figure 3 for a signal structure).

Since the biological function of a given signal structure class is based on the common characteristic binding site of its instances, signal structures belonging to the same class are not necessarily identical but *very similar* to each other. They can slightly vary in their topology, in the sizes of their structure elements, and in their primary structures (see, e.g., Figure 4 for instances of the Iron Responsive Element signal structure class).

5.2. The learning task and data

In this field of research, one of the challenging problems for computational biology is to *detect* instances of known signal structure classes on uncharacterized mRNA. In order to

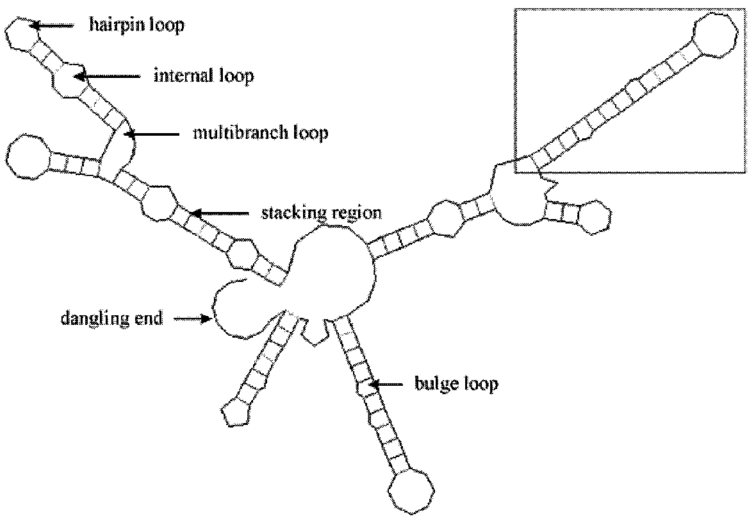


Figure 3. mRNA structure with marked structure elements.

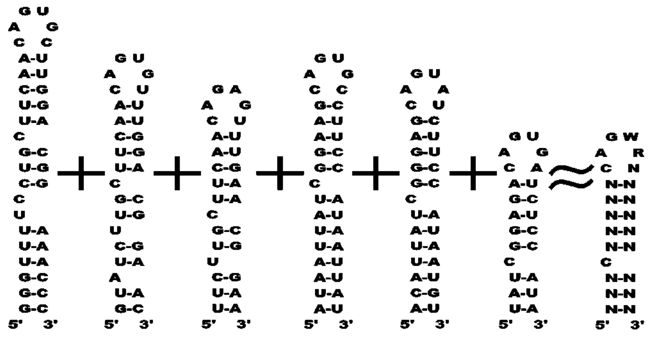


Figure 4. Example for IRE signal structure instances and the corresponding consensus structure described in the IUB-IUPAC code.

solve this problem, one has to consider the related problem of *predicting* the class of an uncharacterized signal structure. That is, we are *given*

– a set of signal structures E, each identified by a single attribute of type `object`, and background knowledge $\mathcal{B}$ describing the structure of each instance with structure elements as determined by biological analysis software,
– each signal structure in E is associated with a target value representing its signal structure class,
– a set of uncharacterized (i.e., without class information) signal structures E_{new}, each described in the same way as E.

The aim is to *find* a prediction of the signal structure class (target value) for each $e \in E_{new}$.

For our experiments, two different example sets were used. The first example set consists of 66 signal structures, composed of 15 IRE (Iron Responsive Element) (Klausner, Rouault & Harford, 1993), 15 TAR (Trans Activating Region) (McCarthy & Kollmus, 1995), 2×15 SECIS (Selenocysteine Insertion Sequence) (Low & Berry, 1996), and 6 histone stemloops (Belasco & Brawerman, 1993). This example set was used for detailed comparative experiments with different representations and background knowledge. To confirm the scalability of the accuracy figures that were obtained, a second experiment was carried out on a larger example set containing 400 signal structures by using the term representation (Bohnebeck et al., 1998b).

5.3. *Background knowledge and representations*

As mentioned above, training and unseen signal structure instances were described in each representation by a single attribute of type `object` pointing to facts on structure information in the background knowledge. Thus, the difference between the experiments is in the complexity and representation of the background knowledge.

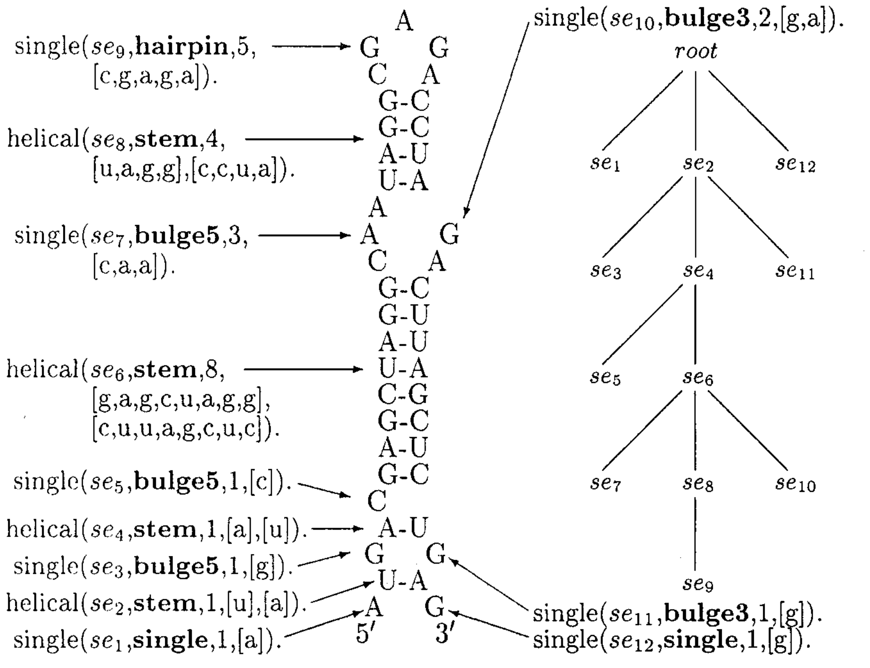
5.3.1. Flat representations. In these representations we have used neither lists, nor terms. The most simple description of signal structures is the *chain* representation, in which they are considered as simple sequences of structure elements. Depending on its type (i.e., is it a *single* sequence, or a *helical* consisting of two attached sequences), the ith structure element of a signal structure is described by a background fact of the form

<se_type>(*struct_id*, i, *size*, *bases*) or
<se_type>(*struct_id*, i, *size*, *bases*, *bases*)

where <se_type> is a background predicate symbol from the set {stem, hairpin, bulge5, bulge3, single} defining the type of the ith structure element (i is of type `number`). Argument *size* of type `number` stores the length of the primary structure of the corresponding structure element. Finally, *bases* of type `discrete` is a special symbol representing the nucleotid sequence of the ith structure element (i.e., the whole sequence is coded by a single symbol in this representation).

The other flat representation we investigated is the *neighbor* represent, which is the *chain* representation extended by additional information on the neighborhoods of structure elements: for each element, both neighboring elements were represented explicitly.

5.3.2. Representations with lists. The two representations used here are basically the same as in the previous case, but instead of type `discrete`, we used type `list` (over the alphabet $\{A, C, G, U\}$) for describing sequences in arguments *bases*.


single(se_9,**hairpin**,5, [c,g,a,g,a]).
helical(se_8,**stem**,4, [u,a,g,g],[c,c,u,a]).
single(se_7,**bulge5**,3, [c,a,a]).
helical(se_6,**stem**,8, [g,a,g,c,u,a,g,g], [c,u,u,a,g,c,u,c]).
single(se_5,**bulge5**,1,[c]).
helical(se_4,**stem**,1,[a],[u]).
single(se_3,**bulge5**,1,[g]).
helical(se_2,**stem**,1,[u],[a]).
single(se_1,**single**,1,[a]).
single(se_{10},**bulge3**,2,[g,a]).
single(se_{11},**bulge3**,1,[g]).
single(se_{12},**single**,1,[g]).
root
se_1
se_2
se_{12}
se_3
se_4
se_{11}
se_5
se_6
se_7
se_8
se_{10}
se_9
5′
3′

secis$_1$(s_1).
structure(s_1, $root$(se_1, se_2(se_3, se_4(se_5, se_6(se_7, se_8(se_9), se_{10})), se_{11}), se_{12})).

Figure 5. The representation and the tree structure of a SECIS signal structure.

5.3.3. Representation with terms. In this representation (see also Figure 5) we described the tree structure of signal structures by using background predicates of the form

structure(*struct_id, topology*)
single(*se_id, type, size, bases*)
helical(*se_id, type, size, bases, bases*).

Argument *topology* of type `term` is the tree structure of the signal structure identified by *struct_id*. Its nodes are structure elements identified by the argument *se_id*, and are further described by the last two predicates. Depending on its type, a structure element is represented by either the *single*/4 or the *helical*/5 predicate. Their second argument of type `constant` is used to describe the type of the structure element identified by argument *se_id* (this information was stored by using different predicate symbols in the flat representations). Finally, the last arguments *size*, and *bases* are the same as in the case of representations with lists.

5.4. *Defining δ for the different types*

In this particular application, we have used a manual definition of δ for constants (referred to as δ_{constant} in the following), the trivial edit costs for the list alphabet $\{A, C, G, U\}$, and the recursively computed editing cost as defined in Section 4 for the nodes of terms. The values of δ_{constant} were as follows:

	$\perp$	single	bulge3	bulge5	hairpin	stem
$\perp$	0	0.03	0.03	0.03	0.90	0.03
single		0	0.03	0.03	0.18	1.0
bulge3			0	0.06	0.12	1.0
bulge5				0	0.12	1.0
hairpin					0	1.0
stem						0

These are heuristic costs and still under development. Two properties are essential. First, helical and single-stranded regions are morphologically substantially different and should therefore not be mapped. Secondly the hairpin loop takes up an exceptional position because its insertion or deletion produces also a quite substantial morphological difference.

5.5. Results and discussion

In order to compare the different levels of representation and background knowledge, experiments were performed using the smaller of the two example sets. Each experiment was performed using leave-one-out cross-validation, determining the best value of k on the training instances, and then classifying the held-out instance. The following table shows the results, where times were taken on a Sun Ultrasparc with OS Solaris 2.5.

background knowledge/ representation	depth	accuracy (%)	k	cpu time (sec)
chain	1	77.2	1	0.5
chain with lists	1	78.8	1	4.3
neighbor	2	83.3	3	2.8
neighbor with lists	2	83.3	1	15.5
term	1	**95.4**	1	32.8

As can be seen in this table, making the list and term background knowledge available has had significant influence on the obtained accuracy, ultimately reaching a level which can be considered as a success in the domain (Bohnebeck et al., 1998b). To confirm the scalability of this result, the final experiment was repeated in the larger example set containing 400 signal structures, yielding a similar level of accuracy:

background knowledge/ representation	depth	accuracy (%)	k	cpu time (sec)
term	1	**96.2**	3	9861.0

As for differences between experiments, besides the remarkable overall increase in accuracy, we see that in this particular domain, the information contained in the base sequences represented by lists did not produce a marked difference in accuracy. To understand why this is so, recall that in the *chain* representations a structure was basically considered as a linear ordered set of its structure elements. The linear ordering was described by an attribute of type `number` thus not supporting the direct reference to a next or previous structure element. This was improved by the *neighbor* representation. The whole topology was described adequately by the *term* representation. In other words, as suspected by the domain experts, the mRNA signal structure problem appears to be inherently structural in nature, and the topology of the molecule, which in RIBL 2.0 can easily be represented using terms, seems to play an important role in prediction.

The experiments also show a marked increase in computation time when terms are used. This is due to the optimizing nature of the edit distance measure which considers the optimal mapping of one term to another. Clearly, this leads to very effective use of the information contained in terms, but may mean that running time becomes problematic for very large applications. This is thus one of the concerns for future work (order-of-magnitude optimizations appear possible e.g. through the use of distance caching), and we are planning to work on a re-implementation of the system.

6. Related work

As an IBL system, RIBL is of course related to the large body of work on instance-based learning, e.g., Aha et al. (1991), Salzberg (1991), Wettschereck and Dietterich (1995). For all variants of IBL that are exclusively based on similarity measures, these variants could be used as a basis for RIBL without problems. For feature weighting approaches (Wettschereck et al., 1997), as pointed out above, the more powerful representation of RIBL requires different weighting schemes, which have been developed by Emde and Wettschereck (1996). Finally, IBL approaches that rely on generalized prototypes or generalized regions in instance space (Salzberg, 1991) do not carry over directly, since in a first-order space, different generalization operators would have to be used. Of course, it would be an interesting topic of further work to see whether for example restricted first-order LGGs (least-general generalizations) are suitable for this purpose.

As for the distance measure used in RIBL, it is most closely related to the recursive computations used in the KBG system (Bisson, 1992a); there, however, they were not used as the basis of an instance-based learning approach, but rather to select objects for generalization. Furthermore, RIBL's similarity measure can be regarded as a generalization of propositional similarity measures, which is not the case in Bisson (1992a).

Recently, several other measures for first-order distances have appeared which we briefly want to contrast with RIBL's measure. Two different ways are distinguished in the literature of distance definitions for first-order logic: *semantical* (Sebag, 1997) and *syntactical* (Hutchinson, 1997; Nienhuys-Cheng, 1997; Nienhuys-Cheng, 1998; Ramon & Bruynooghe, 1998) distances. A semantical distance maps the first-order space into another space by redescribing it in terms of a number of given or learned elementary concepts. Thus, the form of description of the instance is not directly used in similarity computation.

RIBL belongs to the second direction, the syntactical measures, which compute distances directly from the representation of instances. In contrast to RIBL, where distance between atoms is recursively computed based on other background knowledge atoms, most other syntactical first-order distances are based on introducing an appropriate basic metric or pseudo-metric on ground expressions (i.e., on terms and atoms), and then using this basic metric to induce a metric on clauses. Thus, it appears useful to compare such basic metrics on terms with RIBL's term distance metric which is based on edit distances.

6.1. Distances on terms

First, we discuss the relevant results on distances between expressions. By using simple examples, we also show that the behavior of the term editing distance differs fundamentally from the term distances of Hutchinson (1997), Nienhuys-Cheng (1997).

In Hutchinson (1997), a family of pseudo-metrics between terms t_1 and t_2 is defined by

$$D_1(t_1, t_2) = S(\theta_1) + S(\theta_2)$$

where θ_i is the *most general unifier* of t_i and $lgg(t_1, t_2)$ (i.e., the *least general generalization* (Plotkin, 1970)) for $i = 1, 2$, and S is a *size* function on substitutions (see Hutchinson, 1997 for the details).

For two terms with small edit distance, the pseudo-distance with the size function S on substitutions given in Hutchinson (1997) may be unreasonably large. Another property of this pseudo-metric is that the distance between the standard term representations $(n : s^n(\bot))$ of two different natural numbers is always a constant depending on S, while it is a function of n and m, in our case (see Section 6.1.1).

In Nienhuys-Cheng (1997), the 1-bounded recursive metric D_2 is defined as follows:

$$D_2(f(s_1, \ldots, s_n), g(t_1, \ldots, t_m)) = \begin{cases} 0 & \text{if } f = g, n = m \text{ and } t_i = s_i \quad \text{for } i = 1, \ldots, n \\ \dfrac{1}{2n}\displaystyle\sum_{i=1}^{n} D_2(s_i, t_i) & \text{if } f = g \text{ and } n = m\ s_i \neq t_i \quad \text{for some } 1 \leq i \leq n \\ 1 & \text{otherwise} \end{cases}$$

The idea is that the positions of the differences are implicitly weighted (i.e., distance on lower position has higher weight). Considering the term representations of lists, (Nienhuys-Cheng, 1997) does not take into account the *suffix* of the longer list, i.e., the distance between l_1 and l_2 is equal to the distance between l_1 and $l_2 + l$, for all lists l_1, l_2, l with $|l_1| \leq |l_2|$. Furthermore, it may happen that terms with big and other terms with small edit distances may have very close distances by Nienhuys-Cheng (1997). In the case of distance between different natural numbers, the result is independent of the larger number (see Section 6.1.1).

6.1.1. Quantitative comparison of metrics on examples. Since D_1 is a pseudo-metric with no upper bound, and D_2 is a 1-bounded metric on terms, we compare D_1 and D_2 with EDITDIST$_{\text{term}}$ and DIST$_{\text{term}}$, respectively. We consider the following three cases:

Table 2. Comparison of different distances on terms.

Terms	D_1	$\text{EDITDIST}_{\text{term}}$	D_2	$\text{DIST}_{\text{term}}$
$(t_{1,1}, t_{1,2})$	$2n+1$	2	$\frac{1+2^{5-2n}}{3}$	$\frac{2}{2n-1}$
$(t_{2,1}, t_{2,2})$	2	n	$\frac{1+2^{3-2n}}{3}$	$\frac{n}{2n-1}$
$(t_{3,1}, t_{3,2})$	2	$n-m$	2^{-m}	$\frac{n-m}{n+1}$

1. Let $f, a_1, \ldots, a_n$ $(n > 1)$ be different symbols and consider the terms

$$\begin{aligned} t_{1,1} &= f(a_1,\ f(a_2, \cdots f(a_{n-1}, a_n)\cdots) \\ t_{1,2} &= f(a_2,\ f(a_3, \cdots f(a_{n-1}, a_n)\cdots). \end{aligned}$$

2. Let f, a_1, a_2 be different symbols and consider the terms

$$\begin{aligned} t_{2,1} &= f(a_1,\ f(a_1, \cdots f(a_1, a_1)\cdots) \\ t_{2,2} &= f(a_2,\ f(a_2, \cdots f(a_2, a_2)\cdots) \end{aligned}$$

with $|t_{2,1}| = |t_{2,2}| = 2n - 1$ $(n > 1)$.

3. Let $s^n(\perp)$ denote the n applications of the unary function symbol s (successor) to $\perp$, i.e., $s^n(\perp)$ denotes the integer n $(n > 0)$. Let $t_{3,1} = s^n(\perp)$ and $t_{3,2} = s^m(\perp)$ for some $n > m \geq 0$ (all the distances are 0 for $n = m$).

Table 2 contains a comparison of the different term metrics on the above terms (for D_1 we used the size function S of Hutchinson (1997) with $wt_\perp = 0$ and $wt_{a_i} = 1$ for $i = 1, \ldots, n$ and the trivial metrics for the edit distances).

6.2. Distances between sets of literals

One of the most frequently used extension of a distance between expressions to a distance between sets of expressions is the Hausdorff-metric (Hutchinson, 1997; Nienhuys-Cheng, 1997) defining the distance between sets S_1, and S_2 by

$$\max\left(\max_{x\in S_1}\min_{y\in S_2}\text{DIST}(x,y),\ \max_{x\in S_2}\min_{y\in S_1}\text{DIST}(x,y)\right),$$

where DIST is a metric. Although it can be computed in $O(|S_1| \cdot |S_2|)$ for finite sets, and remains a metric, by reducing the distance between sets to a distance between two chosen points it "loses" the information on the whole sets.

Following the approach of Eiter and Mannila (1997), the distance between two sets of atoms is considered as a *weighted matching* problem in Ramon and Bruynooghe (1998), where the weights are defined by the distance between the literals. Although this distance has a number of nice properties, its complexity is cubic in contrast to RIBL's quadratic complexity (see Lines 17–18 in Algorithm 1).

7. Conclusion

In this paper, we have presented the most recent version of the relational instance-based learning system RIBL (RIBL 2.0). After giving a detailed reconstruction of the original RIBL algorithm and similarity measure which was limited to function-free representations, we presented an enhanced similarity measure that can directly handle lists and terms in instance descriptions. As shown by experiments in the domain of mRNA signal structure classification, significant additional power is gained by adding list and term descriptions to the background knowledge. Lists and terms have also proven useful in experiments in other domains, such as part-of-speech tagging for the Hungarian language (Horváth et al., 1999).

While these experiments, and previous experiments with the first version of RIBL, indicate that RIBLs similarity measure is general enough to produce excellent results on a variety of domains, we must of course emphasize that there can be no "best" similarity measure across all problems and goals, so that in other situations, other similarity measures might be preferable. For example, in contrast to Hutchinson (1997), Nienhuys-Cheng (1997), RIBL's distance measure is *stable*, meaning that small changes on a term do not cause big distances. This is usually desirable, but may not always be appropriate. Finally, as with most ILP systems, learning success for RIBL also depends on how background knowledge is represented; here, the addition of lists and terms increases flexibility and allows the user to better fit the background knowledge to the problem domain.

As for future work, above we have already remarked upon further potential for complexity optimization. Firstly, RIBL 2.0 currently uses a non-optimized implementation that does not use indexing nor caching, so that order of magnitude improvements appear possible by eliminating recomputation of previously computed distances. Secondly, further improvements are possible within the algorithm for similarity computation which currently checks all literal matches to find the best partner for each literal. Here it would be possible to use a stochastic approach instead, along the lines of the stochastic matching principle introduced by Sebag and Rouveirol (1997), Sebag (1998). Thirdly, the size of an optimal mapping between terms t_1, and t_2 is currently computed in RIBL by $\text{EDITDIST}_{\text{term}}(t_1/a, t_2/b)$ using the trivial metric on $\{\perp, a, b\}$. As this is a special case of Theorem 6, it might be possible to find a faster algorithm here.

Furthermore, there are some areas where it appears attractive to further enhance the current similarity measure. Firstly, throughout this paper terms have been considered as labeled ordered trees, i.e., the arities of the function symbols (labels) have not been taken into account. In certain situations, it might be desirable to be able to differentiate between different arities or even automatically extend a given distance measure to varying arities. Secondly, in this paper, lists and terms have been defined above alphabets consisting of only *constants*. It appears interesting to extend the similarity metric to *arbitrary* lists and terms, i.e., where the alphabet belongs to an *aggregate* type, so that non-constant functors would also be allowed.

Acknowledgments

We would like to thank Werner Emde and Dietrich Wettschereck for allowing work to continue on RIBL. We are grateful to Tobias Scheffer for proofreading and commentary and to Gábor Keresztes and Joachim Selbig for discussions on the application. We thank Mathias Kirsten for technical support of the implementation. Uta Bohnebeck's biology colleagues in Bremen, Werner Sälter and Dietmar Blohm, were essential to the mRNA application. The first and the second authors were partially supported by the ESPRIT IV Long Term Research Project ILP II (No. 20237).

Notes

1. In RIBL, examples are actually represented by first-order atoms, and the target value can be provided as an additional argument of this atom, or by using several different example predicates representing different classes. We omit these variants for simplicity.
2. Some authors (e.g. Aho, 1990) use different definitions for the edit and for the Levenshtein distances, respectively.
3. The most frequently used structure elements are *stacking regions*, *hairpin loops*, *internal loops*, *bulge loops*, *multibranch loops*, and *dangling ends* (Zuker, 1989).

References

Aha, D. (1997). *Lazy learning*. Dordrecht: Kluwer Academic Publishers.

Aha, D., Kibler, D., & Albert, M. (1991). Instance-based learning algorithms. *Machine Learning*, *6*(1), 37–66.

Aho, A. (1990). Algorithms for finding patterns in strings. In J. van Leeuwen (Ed.), *Handbook of theoretical computer science, algorithms and complexity*, (Vol. A). Amsterdam: Elsevier.

Belasco, J. G. & Brawerman G. (1993). *Control of Messenger RNA Stability*. Oxford: Academic Press.

Bisson, G. (1992a). Conceptual clustering in a first-order logic representation. In *Proceedings of the Tenth European Conference on Artificial Intelligence* (pp. 458–462). Chichester: John Wiley and Sons.

Bisson, G. (1992b). Learning in FOL with a similarity measure. In *Proceedings of the Tenth National Conference on Artificial Intelligence* (pp. 82–87). Cambridge: The MIT Press.

Bohnebeck, U., Horváth, T., & Wrobel, S. (1998a). Term comparisons in first-order similarity measures. In *Proceedings of the Eighth International Conference on Inductive Logic Programming*, (pp. 65–79). Berlin: Springer. Vol. 1446 of *Lecture Notes in Artificial Intelligence*.

Bohnebeck, U., Sälter, W., Horváth, T., Wrobel, S., & Blohm, D. (1998b). Measuring similarity of RNA structures by relational instance-based learning: A first step toward detecting RNA signal structures in silico. In *Proceedings of the German Conference on Bioinformatics*. Technical Report, Universität Köln.

Džeroski, S., Schulze-Kremer, S., Heidtke, K., Siems, K., & Wettschereck, D. (1996). Diterpene structure elucidation from ^{13}C NMR spectra with machine learning. In N. Lavrač, E. Keravnou, & B. Zupan (Eds.), *Intelligent data analysis in medicine and pharmacology*, Dordrecht: Kluwer Academic Publishers.

Eiter, T. & Mannila, H. (1997). Distance measures for point sets and their computation. *Acta Informatica*, *34*(2), 109–133.

Emde, W. & Wettschereck, D. (1996). Relational instance-based learning. In *Proceedings of the Thirteenth International Conference on Machine Learning* (pp. 122–130). San Mateo: Morgan Kaufmann.

Horváth, T., Alexin, Z., Gyimóthy, T., & Wrobel, S. (1999). Application of different learning methods to Hungarian part-of-speech tagging. In *Proceedings of the Ninth International Workshop on Inductive Logic Programming*, (pp. 128–139). Berlin: Springer. Vol. 1634 of *Lecture Notes in Artificial Intelligence*.

Hutchinson, A. (1997). Metrics on terms and clauses. In *Proceedings of the Ninth European Conference on Machine Learning*, (pp. 138–145). Berlin: Springer. Vol. 1224 of *Lecture Notes in Artificial Intelligence*.

Klausner, R. D., Rouault, T. A., & Harford J. B. (1993). Regulating the fate of mRNA: The control of cellular iron metabolism. *Cell*, *72*, 19–28.

Low, S. L. & Berry, M. J. (1996). Knowing when not to stop: Selenocysteine incorporation in eukaryotes. *Trends in Biochemistry Sciences*, *21*, 203–208.

McCarthy, J. E. G. & Kollmus, H. (1995). Cytoplasmic mRNA-protein interactions in eukaryotic gene expression. *Trends in Biochemistry Sciences*, *20*, 191–197.

Muggleton, S. & De Raedt, L. (1994). Inductive logic programming: Theory and methods. *Journal of Logic Programming*, *19/20*, 629–679.

Nienhuys-Cheng, S.-H. (1997). Distance Between Herbrand Interpretations: A measure for approximations to a target concept. In *Proceedings of the Seventh International Workshop on Inductive Logic Programming*, (pp. 213–226). Berlin: Springer. Vol. 1297 of *Lecture Notes in Artificial Intelligence*.

Nienhuys-Cheng, S.-H. (1998). Distances and limits on Herbrand interpretations. In *Proceedings of the Eighth International Conference on Inductive Logic Programming*, (pp. 250–260). Berlin: Springer. Vol. 1446 of *Lecture Notes in Artificial Intelligence*.

Plotkin, G. (1970). A note on inductive generalization. In B. Meltzer & D. Michie (Eds.), *Machine intelligence*, (Vol. 5). Edinburgh University Press.

Ramon, J. & Bruynooghe, M. (1998). A framework for defining distances between first-order logic objects. In *Proceedings of the Eighth International Conference on Inductive Logic Programming*, (pp. 271–280). Berlin: Springer. Vol. 1446 of *Lecture Notes in Artificial Intelligence*.

Salzberg, S. (1991). A nearest hyperrectangle learning method. *Machine Learning*, *6*, 251–276.

Sebag, M. (1997). Distance induction in first order logic. In *Proceedings of the Seventh International Workshop on Inductive Logic Programming*, (pp. 264–272). Berlin: Springer. Vol. 1297 of *Lecture Notes in Artificial Intelligence*.

Sebag, M. (1998). A Stochastic Simple Similarity. In *Proceedings of the Eighth International Conference on Inductive Logic Programming*, (pp. 95–105). Berlin: Springer. Vol. 1446 of *Lecture Notes in Artificial Intelligence*.

Sebag, M. & Rouveirol, C. (1997). Tractable induction and classification in first order logic via stochastic matching. In *Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence*. (pp. 888–893). San Francisco: Morgan Kaufmann.

Shapiro, B. A. & Zhang, K. (1990). Comparing multiple RNA secondary structures using tree comparisons. *Computer Applications in Biosciences*, *6*(4), 309–318.

Tai, K.-C. (1979). The tree-to-tree correction problem. *Journal of the ACM*, *26*(3), 422–433.

Ukkonen, E. (1985). Algorithms for approximate string matching. *Information and Control*, *64*, 100–118.

Wagner, R. A. & Fischer, M. J. (1974). The string-to-string correction problem. *Journal of the ACM*, *21*(1), 168–173.

Wettschereck, D. & Dietterich, T. G. (1995). An experimental comparison of the nearest-neighbor and nearest-hyperrectangle algorithms. *Machine Learning*, *19*(1), 5–27.

Wettschereck, D., Mohri, T., & Aha, D. W. (1997). A review and comparative evaluation of feature weighting methods for lazy learning algorithms. *AI Review Journal*, *11*, 273–314.

Wrobel, S. (1996). Inductive logic programming. In G. Brewka (Ed.), *Advances in knowledge representation and reasoning*. Studies in Logic, Language and Information. Stanford, CSLI-Publishers.

Zhang, K. & Shasha, D. (1989). Simple fast algorithms for the editing distance between trees and related problems. *SIAM Journal on Computing*, *18*(6), 1245–1262.

Zuker, M. (1989). On finding all suboptimal foldings of an RNA Molecule, *Science 244*, 48–52.

Received March 31, 1999
Accepted September 8, 1999
Final manuscript June 11, 2000