Applications of inductive logic programming

Sašo Džeroski (1,2) and Ivan Bratko (1,3)

(1) Jožef Stefan Institute, Jamova 39, 61111 Ljubljana, Slovenia
(2) German National Research Center for Information Technology
    Schloss Birlinbogen, 53745 Sankt Augustin, Germany
(3) Faculty of Electrical Engineering and Computer Science
    University of Ljubljana, Tržaška 25, 61111 Ljubljana, Slovenia

Abstract. A wide variety of applications of inductive logic programming (ILP) have been addressed within the ESPRIT III Project 6020 Inductive Logic Programming. We give a brief overview of these applications, describing in more detail applications that are closer to practice. These include applications in environmental monitoring, mechanical engineering and molecular biology.

1 Introduction

A variety of ILP applications of different degree of difficulty and importance have been addressed within the ESPRIT III Project 6020 Inductive Logic Programming: overviews can be found in [9, 12, 13, 27, 44, 39]. Before giving a general overview and summarizing several of these applications, let us mention that simple or toy problems used to illustrate the workings of ILP systems are here not considered applications. For example, the problem of learning the concept of illegal positions in the KRK chess endgame [52], which has become a standard benchmark, is not considered an application domain.

The ILP applications considered in this paper can be divided into two groups, which we will call proof-of-the-principle applications and real-world applications, respectively. Applications of the first group mainly belong to the artificial realm, but bear strong relevance and resemblance to important practical problems. For example, while the problem of pole-balancing [64] is not of immediate practical interest, it is similar to the problem of controlling a container crane [65], which in turn is of immediate practical significance. A typical proof-of-the-principle application formulates a generic task as an ILP problem and then addresses an illustrative instance of the generic task which is simpler than real-world instances of the same generic task.

For the second group of applications, real-world data are available, as well as a domain expert who shows a strong interest for solving the problems at hand with ILP. This expert inspects and evaluates the obtained results. Thus, real-world domains are characterized by an interaction between ILP practitioners and a domain expert. For instance, the problem of predicting protein secondary structure [55] is of immediate practical importance as the shape (secondary structure) of proteins determines their function. Furthermore, two domain experts (R.D. King and M.J.E. Sternberg) were involved in the application of the ILP system GOLEM [54] to this problem.
The application domains addressed within the project include software engineering [4, 10], modeling and control of (dynamic) systems [14, 9], natural language processing [21], knowledge discovery in databases [27], database design [22, 34], data analysis [48, 26], medical diagnosis [46], environmental monitoring [30], mechanical engineering [24, 44, 42, 56], and molecular biology [55, 38, 61]. The boundary between toy problems and proof-of-the-principle applications, as well as between the latter and real-world applications, is not clear cut. It is thus sometimes difficult to judge which class a particular ILP application belongs to. Nevertheless, the above list of application domains is ordered so that it progresses from proof-of-the-principle to real-world applications.

The remainder of the paper gives a brief overview of the applications of ILP in the areas of software engineering, modeling and control of (dynamic) systems and more detailed accounts of applications in the areas of environmental monitoring, mechanical engineering, and molecular biology. In particular, results in the domain of biological classification of river water quality, finite element mesh-design, and predicting protein secondary structure are described. Short summaries of results on predicting drug activity and mutagenicity are also given. A list of ILP applications within and outside the ESPRIT III Project 6020 Inductive Logic Programming not mentioned in the overview is also given. We conclude with a discussion of the relevance of current ILP applications for the respective application domains and for ILP research.

2 Software engineering

Several problems within the area of software engineering have been addressed by ILP researchers. These include: program development from high level specifications through data reification, program construction using a strong declarative bias, construction of invariants for proving program correctness, and generation of test cases. In all cases, illustrative examples have been worked out in detail to explain the underlying principles.

2.1 Program construction

In program construction from higher order specifications, functions in the specification language (higher level) are to be implemented in the target language (lower level). Thereby abstract data types at the higher level are to be reified into concrete data types at the target language level. For example, sets can be reified into lists. In [10] this refinement problem is formulated in the ILP framework. As an illustration, the ILP system MARKUS [35] was used to automatically construct a program that implements the set union operation (higher level) as list concatenation (lower level).

A different approach to program construction is taken by Bergadano and Gunetti [2, 4, 5, 6]. They use a strong declarative bias instead of a higher level specification. This bias specifies a set of clauses that can appear in the target program, hence the name clause sets for the bias formalism used. The task is to select a subset of a given clause set that is consistent and complete with respect to a given set of examples, which are ground facts of several target predicates.

The first system that uses the clause sets bias is FILP [2]. FILP evaluates clauses from the clause set independently of each other, using an extensional notion of coverage. This implies a need for complete examples of the target predicates when learning recursive clauses. FILP uses queries to achieve this completeness.

TRACY [4] and TRACYnot [5] avoid the need for queries and learn definite, re-
spectively normal, logic programs from clause sets and examples. The basic idea is to treat a clause set as a logic program and try to prove the positive examples from it. The clauses used in the proofs form a candidate hypothesis (obviously, they are not evaluated independently of each other - they are evaluated intensionally). If such a candidate hypothesis covers negative examples, backtracking occurs.

TRACY and TRACY\textsuperscript{not} generate a definite, respectively normal, logic program which is consistent and complete with respect to the given set of examples when one exists within the clause set. They have been applied to induce slightly larger logic programs than those considered by most ILP systems, including a program for finding a hamiltonian circle in a graph [5]. A more detailed treatment of software engineering applications of ILP, including inductive test case generation, can be found in [6].

2.2 Inducing invariants for program verification

In formally proving the correctness of procedural programs, we need to find suitable preconditions of program statements. Such a precondition has to be sufficiently strong to imply the truth of the postcondition of the statement, and at the same time sufficiently weak so that it can be proved to hold at all times of program execution.

Of particular interest is the problem of finding suitable conditions that are true inside program loops, called loop invariants. Once such a suitable loop invariant is known, it remains to prove the corresponding theorems ensuring that the invariant is in fact always true. In general, the more difficult task is the construction (usually guessing) of such suitable invariant conditions.

The idea explored by Bratko and Grobelnik [10] is that ILP can be used for constructing loop invariants. The procedural program that is to be proved correct can be executed, and the resulting execution traces used as examples. The states of program variables at a given point in the program represent positive examples for the condition associated with that point in the program. The remaining questions are how to obtain negative examples, and what should the background knowledge be.

Negative examples can be generated by using existing variable dependencies and a controlled closed-world assumption. Namely, it can be shown that some program variables are functions of the other variables. The values of the dependent variables are determined by the values of the independent variables. Changing the value of a dependent variable in a given positive example then inevitably creates a negative example.

Bratko and Grobelnik [10] give as an example the program for integer division by iterative addition and subtraction. Its loop invariant was successfully reconstructed by several ILP systems. They also induced invariants for a parallel program [11]. These are used for proving important properties, such as safety (a parallel program is safe if all critical sections are protected against simultaneous execution).

2.3 Inductive test case generation

Bergadano et al. [3, 6] apply ILP to generate a set of cases for alternative-based testing of programs. In addition to $P$, the program to be tested, a set $P$ of alternative programs is given. A set of test cases is generated that is adequate in the sense that it distinguishes $P$ from all alternatives. If at least one of the given programs is correct, the obtained test set will also be reliable, i.e., it will reveal any errors that may be present in $P$. 
Test cases are generated iteratively. Given a partial set of cases, i.e., input-output pairs, a program $P'$ from $\mathcal{P}$ consistent with this partial set is induced. An input-output pair where $P$ and $P'$ differ is next chosen, if possible (otherwise $P'$ removed from $\mathcal{P}$), and the procedure repeated. If no program from $\mathcal{P}$ consistent with the current partial set of cases can be induced, the procedure is terminated, as the set of cases is adequate.

3 Modeling, control and design

This section summarizes the applications of ILP approaches to problems related to dynamical systems and qualitative physics. We first describe the task of qualitative identification of dynamical systems or, in other words, inducing a qualitative model from an example qualitative behavior. We then describe the synthesis of control rules from examples of successful control, termed behavioral cloning. We finally summarize the attempt to design devices "from first principles", given examples of the desired behavior and qualitative physics specifications of the components available.

3.1 Learning qualitative models

A fundamental problem in the theory of dynamic systems is the identification problem, defined as follows: given examples of the behavior of a dynamical system, find a model that explains these examples. Motivated by the hypothesis that it should be easier to learn qualitative than quantitative models, Bratko et al. [14] formulated the qualitative identification problem as an ILP problem. In their work, models are sets of Qualitative Differential Equations (QDEs) that constrain the values of system variables. QDEs are a widely known formalisms for defining qualitative models of dynamical systems [43].

A Prolog implementation of QDE constraints is provided as background knowledge. Example behaviors of the modeled system are used to generate positive training examples (these are the legal states of the systems that appear in the behaviors), while negative examples are generated as near misses. The general ILP problem formulation becomes thus instantiated as follows: given QDEconstraints and ExampleBehaviors, find a QualitativeModel, such that QDEconstraints and QualitativeModel explain the ExampleBehaviors.

Models of simple dynamical systems have been induced using ILP. A model of the U-tube system has been successfully reconstructed by mFOIL [28]. The ILP system GOLEM constructed a rather different model [14], which is conditionally equivalent to the correct one: if the qualitative simulation starts in a legal qualitative state then the model induced by GOLEM will generate a correct behavior.

3.2 Learning control rules

The task of learning a control rule for a dynamical system from successful control traces can be formulated as follows: given examples of the form $(Time, Action, State)$, find a functional relation between the State and Action of the following form $Action(Time) = f(State(Time - Delay))$ [65]. For simplicity, we will assume no delay, i.e., $Action = f(State)$. The examples are derived from the successful performance of a controller (typically a skilled human operator), where the controlled system is brought from an initial to a goal state through a series of appropriate control actions. This task is often
named behavioral cloning.

Current experiments in behavioral cloning mostly assume the above formulation, where the goal is implicit in the performance trace. The induced controllers can thus be applied only to the very same control task that was used to generate the examples. Obviously, it is desirable to obtain controllers that are applicable to a range of similar tasks where the goal can vary. A formulation of the behavioral cloning task that assumes \( \text{Action} = f(\text{State}, \text{Goal}) \) would facilitate the induction of more flexible controllers.

Another issue that deserves attention is the form of the function \( f \). Most approaches to behavioral cloning assume that \( f \) can be implemented by a propositional set of classification rules, a classification tree or a regression tree. This formulation allows only the use of conditions such as equalities and inequalities between a state variable and a constant (but not between state variables). ILP would facilitate the use of background knowledge and allow a richer expressive formalism for the function \( f \). Given background knowledge \( B \), an ILP formulation of behavioral cloning assumes \( \text{Action} = f_B(\text{State}, \text{Goal}) \) [31].

Džeroski et al. [32] illustrate the above approach by applying ILP to learn to control a well-known dynamical system: the pole-on-cart (inverted pendulum). A piece-wise linear controller for this dynamic system was successfully reconstructed from example traces of successful control. The system DINUS [44] was used, which transforms ILP problems to propositional form and can therefore handle real numbers and predict real-valued class variables. The learned controller works successfully from the initial and goal positions used in training. In addition, experiments indicate that it performs successfully over the whole possible space of initial and goal positions.

The problem of pole-balancing is a well-known benchmark problem, but is not of immediate practical interest. However, it is similar to the problem of controlling a container crane, which in turn is practically important. This task has been addressed by Urbančič and Bratko [65], who use propositional approaches to reconstruct human skills from behavior traces solving a single control problem. Another interesting problem in behavioral cloning is the problem of learning to pilot a flight simulator [47, 59]. As this is a more complex problem, it is divided into stages. Up to now, propositional approaches have been used to construct separate controllers (decision trees) for each stage. Recently, Camacho [16] has induced rules for transitions between successive stages of a fixed flight plan with varying parameters (e.g., goal altitude) using ILP. He also reports on additional experiments in learning to fly using ILP [17], which have had mixed success.

3.3 Design from first principles

Innovative design "from first principles" is concerned with designing devices (or artifacts) that satisfy a given specification of their desired behavior. Bratko [8] formulated the problem of innovative design as an ILP problem. The design process is viewed as the process of structuring available elementary components in such a way that they together realize some specified target behavior. The approach addresses the design from first principles in the sense that the functional behavior of an artifact is derived from the physics of the elementary components available to the designer. The approach proposed involves: specification of the target artifact by examples of its intended behavior, qualitative physics definition of the behavior of the elementary components available, and ILP as the mechanism for conceptually constructing the device. As an illustration,
MARKUS [35] was applied to construct simple electric circuits from examples of their intended behavior and the qualitative physics of simple electrical components.

4 Biological classification of river water quality

Increasing importance is placed on the use of riverine ecology as a means of monitoring and classifying river quality, both in terms of water quality and its broader environmental quality [58]. The various biological flora and fauna, such as attached algae, macrophytes and benthic (or river bed) macro-invertebrates, are seen as continuous monitors of the rivers’ “health”, and field data on these are used to classify the river. In the case of water quality monitoring and classification, the biological methods are used to complement the more traditional chemical methods.

At present, the most suitable single group for monitoring purposes is considered to be the benthic macro-invertebrates. These animals form part of the community associated with the river bed and are relatively immobile. They are present in all rivers, except in cases of extreme pollution, and cover a range of life modes and trophic levels. Taxonomic classification of these organisms to family or genus level is not demanding, and qualitative identification is often carried out at the river bank. Also the different species are known to have different sensitivities to pollutants, thus the structure of the benthic macro-invertebrate community is affected by both degradable organic matter (sewage) and toxic pollutants (pesticides and heavy metals).

The task addressed here is to interpret benthic samples of macro-invertebrates in water quality terms. In other words, given are samples of the river beds at different sites and their classification into one of five quality classes. The task is to learn general rules that will be able to classify new samples [30]. Given are 292 field samples of benthic communities taken from British Midlands rivers and classified by an expert river ecologist (H. A. Hawkes).

Two ILP systems were applied to the problem of classification of biological samples. GOLEM [54] works in the normal ILP setting [53], where the task is to find classification rules that explain the training examples, while CLAUDIEN [23] works in the nonmonotonic ILP setting, where the task is to find valid rules that are confirmed by the training examples. A propositional learning system, CN2 [18, 29] was also applied to learn classification rules.

For the normal ILP setting, the problem was formulated as follows: for each class a separate ILP problem was created, where the positive examples are the samples classified in that class, and all the other samples are negative examples. Thus, the target predicates were $b1a(X), b1b(X), b2(X), b3(X)$, and $b4(X)$. The background knowledge consisted of eighty predicates of the form $\text{family}(X, A)$, each denoting that $\text{family}$ is present in sample $X$ at abundance level $A$. (In several cases, identification was carried to levels other than family, e.g., species or genera level. For simplicity, we will use the term family throughout, regardless of the taxonomic identification level.) Predicates of this kind include $\text{tipulidae}(X, A), \text{asellidae}(X, A)$, etc. In addition, the background predicate $\text{greater\_than}(A, B)$ was available, stating that abundance level $A$ is greater than abundance level $B$.

The default settings of GOLEM were used, except for the fact that rules were allowed to cover up to five negative examples. GOLEM produced three rules for class B1a, fourteen rules for B1b, sixteen for B2, two for B3, and none for B4. For example, the rule $b1a(X) \leftarrow \text{leuctridae}(X, A)$ states that a sample belongs to the best
water quality class if Leuctridae are present. This rule covers forty-three positive and four negative examples, and agrees with expert knowledge; the family Leuctridae is an indicator of good water quality. Another good rule is the following: \( b1b(X) \leftarrow ancelidae(X, A), gammaridae(X, B), \text{hydropsychidae}(X, C), \text{rhyacophilidae}(X, D), \text{greater} \text{than}(B, A), \text{greater} \text{than}(B, D) \). Gammaridae in abundance is a good indicator of class B1b, along with the other families present.

Out of the thirty-five rules, twenty-five are considered good or acceptable by the expert, but some of them are judged to be too specific [30]. Note that GOLEM cannot use the absence of particular families (families) in the rules within the representation adopted here. Together with GOLEM's strategy of looking for the most specific rules consistent with the examples, this may have influenced the generality (specificity) of the generated rules. This may also be the reason that no rules for class B4 were induced. Namely, the absence of most families is characteristic for this class.

Unlike GOLEM, CLAUDIEN [23] checks all possible rules, within a specified language, for consistency with the given example. Only tests of the presence of different families were allowed in the antecedents (no absence tests and no abundance level tests), while one or more quality classes were allowed in the consequents. Rules were required to cover at least thirty examples.

Altogether, seventy-nine rules were generated. Of these, twenty-eight involved the presence of a single family. These rules in fact specify the range of quality classes in which a certain family is present. The rule \( b1a(X) \lor b1b(X) \leftarrow \text{perlodidae}(X, A) \) specifies that Perlodidae are found in good quality water (classes B1a and B1b). If Rhyacophilidae are also present, then the water is of class B1a: \( b1a(X) \leftarrow \text{perlodidae}(X, A), \text{rhyacophilidae}(X, B) \). Both rules are judged by the expert to be good.

Only nine rules had a single class (B1a) in the conclusion. The others had a number of possible classes in the conclusion, which was considered natural and understandable by the expert. This indicates that class B1a is easy to characterize in terms of the families present, while for the other classes references to the abundance levels or absence of certain families is required in order to find significant rules (that cover at least thirty samples). Overall, two thirds of the rules were considered to be good. The rest were not outright wrong, but still unacceptable to the expert for a number of reasons, such as mentioning families which are judged to be irrelevant for a particular water quality class.

In summary, the rules induced by the ILP systems GOLEM and CLAUDIEN were found to be consistent with the expert knowledge to a great extent. Compared to other methods for automating the classification process, such as neural networks, ILP produces symbolic rules that can be used as a knowledge base for an expert system. The rules generated by CLAUDIEN were judged to be the most intuitive and promising. This is due to the coverage of more than one class by a single rule; this is important as the classification problem is based on a discretization of a continuous space. The fact that a single family is used in some rules also contributes to the overall intuitiveness of the CLAUDIEN rules.

5 Finite-element mesh design

The problem of finite-element mesh design arises in numerical computation based on finite-element methods. For example, given an object and forces acting on it, finite-element methods can be used to compute the pressure and deformations throughout the
object. Finite element (FE) methods, however, require that the object is partitioned into finite elements, resulting in a finite element mesh.

Numerical approximations are sufficiently accurate if the elements are sufficiently small. In general, the finer the mesh, the smaller the error. However, a dense mesh results in a lengthy computation. The problem, then, is to find a suitable compromise between the coarseness of the mesh and the accuracy of the computations.

Normally some regions of the object require a denser mesh whereas in other regions a coarser mesh still suffices for good approximation. There is no known general method that would enable automatic determination of optimal, or even reasonably good meshes. However, many examples of successful meshes for particular objects have been accumulated in the practice of FE computations. These meshes can be used as sources of examples for learning about the construction of good meshes.

In general the mesh depends on the geometric properties of the object, the forces acting on it, and the relations between different components of the object. The mesh density in a region of the object depends also on the adjacent regions. Because of these relational dependences, the mesh design problem is a very natural application of ILP.

The application of ILP to the problem of mesh design in mechanical engineering was developed mainly by B. Dolšak, who also served as a domain expert, in collaboration with several ILP researchers [25, 7, 24]. Several ILP systems have been applied in this domain, which has become a standard benchmark. Recent experiments with variants of the same learning data are described for example in [33, 42, 56].

An object to be partitioned is represented as (1) a set of edges, (2) the properties of the edges, and (3) relations among the edges. These properties and relations are represented by background knowledge predicates, such as \textit{short(Edge)}, \textit{loaded(Edge)}, \textit{not.loaded(Edge)}, \textit{two.side.fixed(Edge)}, \textit{neighbor.xy(Edge1, Edge2)}, etc. In experiments to learn a characterization of the density of a mesh in terms of these relations, five to ten meshes known to be numerically adequate are normally used as sources of examples for learning. The target relation to be learned is: \textit{mesh(Edge, N)} where \textit{Edge} is the name of an edge in the structure, and \textit{N} is the recommended number of finite elements along this edge. The available meshes comprise several hundreds of edges. Each edge is used as an example for learning, and typically some additional positive examples are derived from the meshes. So a typical number of examples is between 300 and 600. Negative examples are generated by a kind of closed-world assumption which gives rise to several thousands of negative examples.

Several relational learning algorithms were tried on this data including GOLEM [54], LINUS [45] and FOIL [57]. The results obtained with GOLEM were judged to be the most satisfactory. However, GOLEM generated a large number of rules and many of them were useless. It was fortunately possible to design a set of constraints to distinguish between "good" rules and bad rules. This enabled automatic post-filtering of the induced rule set whereby useless rules were automatically eliminated. In a more recent experiment with the ILP system CLAUDIEN [23] Dolšak (personal communication) reported that it was possible to formulate language bias constraints that only allow good rules to be built. This facility in CLAUDIEN turned out to be very practical and effective. Note also that CLAUDIEN does not need negative examples as it applies a kind of closed-world assumption itself.

The resulting rules were of interest to expert users of the finite element methods. According to their comments, these rules reveal interesting relational dependences. Here we give an interesting example of such a rule (in Prolog syntax).
mesh(Edge, 7) ← usual length(Edge),
neighbor_xy(Edge, EdgeY), two_side_fixed(EdgeY),
neighbor_xz(EdgeZ, Edge), not_loaded(EdgeZ).

This rule says that partitioning Edge into 7 elements is appropriate if Edge has a
neighbor EdgeY in the xy-plane so that EdgeY is fixed at both ends, and Edge has
another neighbor EdgeZ in the xz-plane so that EdgeZ is not loaded.

The following recursive rule was also generated by GOLEM:

mesh(Edge, N) ← equal(Edge, Edge2), mesh(Edge2, N). This observes that an edge's
partition can be determined by looking for an edge of the same length and shape po-

tioned similarly in the same object. In other words, this can be viewed as GOLEM's
discovery that an edge may inherit a suitable partition from similar edges in the struc-
ture. Of course, for this rule to be computationally useful, at least some of the equivalent
edges must have its partition determined by some other rule.

The accuracy of the induced rule sets was investigated in detail in [24]. It is,
however, not so clear what is the best way of estimating the accuracy in this domain.
A straightforward method is cross-validation whereby a number, say 10 %, of all the
available examples (that is edges) are used as a test set, and the remaining examples
are used as a training set. In this approach, the test set accuracy of the rules induced
by GOLEM was (on average) as follows: the rules suggested correct partition of an edge
into finite elements in 78 % of all test cases, incorrect in 2 % of the cases, and 20 % of
the test edges remained undecided (not covered by the induced clauses).

A more practically realistic evaluation situation is that the generated clauses are ap-
plied to determining a mesh for a completely new structure, one not used for learning.
In such an experiment with GOLEM, using the evaluation method "leave-one-structure-
out", 67 % of the edges of a new structure were classified correctly, 22 % incorrectly, and
11 % remained unclassified. Although the error rate here seems rather high, it may
be close to acceptable in the practice of mesh design. Because of some general local
consistency constraints used in mesh generators, many of the errors can be automati-

cally corrected. Another debatable point of these evaluation attempts is that they
do not take into account the amount of error. For example, if the correct partition
is into five elements, recommendation four is considered as an equally serious error as
recommendation one. In future evaluations this should be taken into account.

6 Molecular biology

The ILP applications in the area of molecular biology have come closest to real-world
relevance. Predicting protein secondary structure [55], predicting drug activity through
modeling structure-activity relations [38, 39] and predicting the mutagenicity of aro-
matic and heteroaromatic nitro-compounds [61] are all of immediate practical interest.
In these domains, results that are better or equal to the best previous results have been
obtained, in addition to understandable and relevant new knowledge.

6.1 Predicting protein secondary structure

We now briefly review the problem of learning rules for predicting the secondary struc-
ture of proteins, to which GOLEM has been applied [55]. We first describe the problem,
then the relevant background knowledge, and finally the results.

A protein is basically a string of amino acids (or residues). Predicting the three-
Figure 1: Rules for predicting α-helix secondary structure.

Level 0 rule

\[ \text{alpha0}(A, B) \leftarrow \text{octf}(D, E, F, G, B, H, I, J, K), \]
\[ \text{position}(A, D, Q), \text{not}. p(Q), \text{not}. k(Q), \]
\[ \text{position}(A, E, O), \text{not}. \text{aromatic}(O), \text{small}. \text{or}. \text{polar}(O), \]
\[ \text{position}(A, F, R), \]
\[ \text{position}(A, G, P), \text{not}. \text{aromatic}(P), \]
\[ \text{position}(A, B, C), \text{very}. \text{hydrophobic}(C), \text{not}. \text{aromatic}(C), \]
\[ \text{position}(A, H, M), \text{large}(M), \text{not}. \text{aromatic}(M), \]
\[ \text{position}(A, I, L), \text{hydrophobic}(L), \]
\[ \text{position}(A, K, N), \text{large}(N), \text{ltv}(N, R). \]

Level 1 rule

\[ \text{alpha1}(A, B) \leftarrow \text{octf}(D, E, F, G, B, H, I, J, K), \]
\[ \text{alpha0}(A, F), \text{alpha0}(A, G). \]

Level 2 rule

\[ \text{alpha2}(A, B) \leftarrow \text{octf}(C, D, E, F, B, G, H, I, J), \]
\[ \text{alpha1}(A, B), \text{alpha1}(A, G), \text{alpha1}(A, H). \]

dimensional shape of proteins from their amino acid sequence is widely believed to be one of the hardest unsolved problems in molecular biology. It is also of interest to the pharmaceutical industry since the shape of a protein determines its function.

The sequence of amino acids is called the primary structure of the protein. Spatially, the amino acids are arranged in different patterns (spirals, turns, flat sections, etc.). The three-dimensional spatial shape of a protein is called the secondary structure. When trying to predict the shape (secondary structure) it is easier to consider only one particular shape (pattern), instead of the multitude of possibilities; the α-helix (spiral) shape was considered by Muggleton et al. [55].

The target relation alpha(Protein, Position) specifies that the residue at position Position in protein Protein belongs to an α-helix. Negative examples state all residue positions in particular proteins which are not in an α-helix secondary structure. For instance, the positive example alpha(1HMQ, 104) means that the residue at position 104 in protein 1HMQ (hemerythrin met) is in an α-helix. The negative example alpha(1HMQ, 105) states that the residue at position 105 in the same protein is not in an α-helix (it actually belongs to a β-coil secondary structure).

The following information is used as background knowledge [55]. The relation position(A, B, C) states that the residue of protein A at position B is the amino-acid C. Arithmetic relations allow indexing of the protein sequence, relative to the residue considered. The relation octf(A, B, C, D, E, F, G, H, I) specifies nine adjacent positions in a protein, while the relations alpha.triplet(A, B, C), alpha.pair(A, B) and alpha.pair4(A, B) are defined declaratively as alpha.triplet(n, n + 1, n + 4), alpha.pair(n, n + 3), alpha.pair4(n, n + 4). Physical and chemical properties of individual residues are provided, such as hydrophobicity, hydrophilicity, charge, size, polarity, etc. Relations between the particular values of properties, such as less.than(X, Y), are also provided as background knowledge.
Let us now proceed with a very short description of the experimental setup and the results [55]. Sixteen proteins from the Brookhaven database were used, twelve for training and four for testing. Without addressing in detail how GOLEM was run to induce rules from the training data, let us mention that each of the induced rules was allowed to misclassify up to 10 instances and some subjective statistical criteria were used to judge its significance before allowing it into the set of rules used for classification. The induced rules covered around 60% of the instances.

To improve the coverage of these preliminary rules, the learning process was iterated. The predicted secondary structure positions found using the initial rules (called level 0 rules) were added to the background information. GOLEM was then run again to produce new (level 1) rules. This was necessary as the level 0 predictions were speckled, i.e., only short α-helix sequences were predicted. The level 1 rules in effect filtered the speckled predictions and joined together short sequences of α-helix predictions. The learning process was iterated once more with level 1 predictions added to the background knowledge and level 2 rules were induced. Finally, the symmetric variants of the rules induced at level 1 and level 2 were added to the rule set (e.g., \( \alpha \text{pha1}(A, B) \leftarrow \alpha \text{ctf}(D, E, F, G, B, H, I, J, K), \alpha \text{pha0}(A, F), \alpha \text{pha0}(A, G) \)): this was suggested by a domain expert that inspected the rules.

Applying GOLEM to the training set produced twenty-one level 0 rules, five symmetric level 1 rules and two symmetric level 2 rules. For illustration, Figure 1 gives one rule from each level. The induced rules achieved accuracies of 78% on the training and 81% on the testing set. For comparison, the best previously reported result is 76%, achieved by using a neural network approach [41]. The rules induced by GOLEM also have the advantage over the neural network method of being more understandable.

6.2 Modeling structure-activity relations

A central concern of chemistry is understanding the relationships between chemical structure and activity. In most cases, these relationships cannot be derived solely from physical theory and experimental evidence is essential. Such empirically derived relationships are called Structure Activity Relationships (SARs). In a typical SAR problem, a set of chemicals of known structure and activity are given, and the task is to find a predictive theory relating the structure of a compound to its activity. This relationship can then be used to select structures with high or low activity. Typically, knowledge of such relationships is used for devising clinically effective, non-toxic drugs.

The ILP system GOLEM [54] has been applied to several problems of this kind, including the problem of inhibition of E. Coli Dihydrofolate Reductase by two different groups of drugs (pyrimidines and triazines) [38, 39] and the properties (toxicity, acetylcholinesterase inhibition, etc.) of Tacrine (a drug for treating Alzheimer's disease) derivatives [39]. The ILP formulation of the problems compares the properties of pairs of compounds with known activity. The target relation is therefore \( \text{great}(X, Y) \). For example, in the problem of inhibition of E. Coli Dihydrofolate Reductase a fact \( \text{great}(d_1, d_2) \) specifies that inhibition by drug \( d_1 \) is higher than that by \( d_2 \).

The background knowledge contains predicates specifying the chemical structure of the drugs with known activity. In the case of pyrimidines, the drugs follow a template and differ only by the substitutions at three positions. Thus, each drug is specified by a triple of substituents. Properties of the substituents such as polarity, size, flexibility, etc., are also part of the background knowledge.
For the problem of inhibition of E. Coli Dihydrofolate Reductase, GOLEM induced nine rules comparing drug activities. The Spearman rank correlation of the drug activity order predicted by GOLEM with the actual order is 0.92 on the training set of drugs and 0.46 for the testing set of drugs [38]. For a classical SAR approach, taken by Hansch et al. [36], the Spearman rank correlations are 0.79 and 0.42, respectively. Besides achieving better accuracy than traditional methods, the induced rules also provide a description of the chemical laws governing the problem.

6.3 Predicting mutagenicity

Srinivasan et al. [61] applied the ILP system Progol [51] to induce theories for predicting the mutagenicity of a set of 230 aromatic and heteroaromatic nitro-compounds. The prediction of mutagenicity is important as it is relevant to the understanding and prediction of carcinogenicity. The compounds used in this study are more heterogeneous structurally than those used in the drug design domains and can only be fully represented in a first-order setting.

Of the 230 compounds, 138 have positive levels of log mutagenicity, these are labeled "active" and constitute the positive examples: the remaining 92 compounds are labeled "inactive" and constitute the negative examples. The target relation is in this case active(C), stating that compound C has positive log mutagenicity. The background knowledge contains the structure of the compounds represented as a list of atoms and bonds that can be found in each compound. The predicate atm(C, A, E, T, Charge) states that atom A in compound C is an atom of element E (e.g., carbon), of type T (e.g., aromatic carbon) with charge Charge. The predicate bond(C, A1, A2, BT) states that there is a bond of type BT (e.g., aromatic bond) between atom A1 and atom A2 of compound C. The facts for these predicates were generated by the molecular modeling program QUANTA, where the compounds were entered through a chemical editing facility.

In addition, four attributes are provided for analysis of the compounds. These can be used directly by both propositional and ILP learners. They are: (1) the hydrophobicity of the compound (termed logP); (2) the energy level of the lowest unoccupied molecular orbital (termed LUMO); (3) a boolean attribute identifying compounds with 3 or more benzyl rings (termed indicator variable I1); and (4) a boolean attribute identifying a sub-class of compounds termed acenthryles (termed indicator variable Ia). The last two are pre-selected structural features that incorporate chemical expert knowledge.

Generic structural knowledge was used as background knowledge in some experiments [62, 63] with Progol. It includes definitions of the concepts of methyl groups, nitro groups, aromatic rings, heteroaromatic rings, connected rings, ring length, and the three distinct topological ways to connect three benzene rings. These definitions are generic to the field of organic chemistry.

The 230 compounds are divided into two sets: 188 compounds that could be fitted using linear regression (regression-friendly set), and 42 compounds that could not (regression-unfriendly set). In the Progol experiments, accuracies of theories constructed for the 188 compounds were estimated from a 10-fold cross-validation and the accuracy of theories for the 42 compounds were estimated by a leave-one-out procedure.

In summary, Progol produced theories that perform as well as linear regression or neural networks (88% vs. 89% accuracy) on the regression-friendly set and much better (88% vs. 69% accuracy) on the regression-unfriendly set [62]. In comparison with CART
[15], no significant differences in performance exist when the pre-selected structural features are available. However, when these features are not available, Progol performs much better on the regression-friendly set (88 % vs 83 %). Progol also performed better than FOIL [57], especially with regard to monotonic improvement of performance with increasing amounts of background knowledge [63].

It is worth noting that Progol generated a single rule for the regression-unfriendly set. The structure indicated by this rule is a new structural alert for high mutagenicity in chemical compounds. An expert explanation for this has also been found [62].

7 Discussion

We have reviewed a number of ILP applications addressed within the ESPRIT III Project 6020 Inductive Logic Programming. They vary considerably in the degree of sophistication and significance with respect to the area of application. The applications in the area of molecular biology have come closest to practical importance. The river water quality and mesh design applications are slightly further away from deployment, while the applications in software engineering, modeling, control and design mainly illustrate the principles underlying these application areas. Other applications within the project include early diagnosis of rheumatic diseases [46], analysis of results from discrete event simulations [48], and analysis of Rachmaninoff's piano performance [26]. The MOBAL modeling environment [50] and its ILP component RDT [37] was applied to the problem of security management in telecommunication networks [60]. An improved version of RDT has been used to learn concepts necessary for navigation of autonomous mobile robots [40].

Several ILP applications outside the project deserve to be mentioned here. Cohen [19] used ILP to recover an abstract specification of a large software system. Mooney has been concerned with applying ILP to problems in the area of natural language processing, such as mapping sentences to case-role representations [66] and generating the past tense of English verbs [49]. Cohen [20] has also addressed the task of text categorization using both propositional and ILP approaches. Finally, Amin et al. [1] apply ILP to the recognition of a limited subset of hand-printed Chinese characters.

While none of the ILP applications discussed in this paper is fielded, they are still of considerable importance. First, several applications have generated useful new knowledge in the corresponding domains, especially the applications in molecular biology. For example, knowledge of quality sufficient for publication in high standard scientific journals has been derived in the domains of protein secondary structure prediction and structure-activity relationships.

Second, without careful representation engineering, ILP seems to do better than propositional learners in several application domains, such as predicting mutagenesis and generating the past tense of English verbs. While careful representation engineering may result in features that enable propositional systems to perform just as well as ILP, this process is in principle more demanding for the user.

Third, research issues that are important for practical ILP applications have been identified. These include handling of different types of numerical information and constraints, learning probabilistic concepts, and assessing the relevance of background knowledge predicates. Much more research effort is also needed to enable learning in complex structural domains that involve recursion and require predicate invention.

Finally, several application domains where the use of ILP is expected to be beneficial
have been identified. These are natural language processing, data mining and discovery, design and configuration, and database design. Proof-of-the-principle applications in these areas include: induction of definite clause grammars and formation of diminutives in Dutch [21] (natural language processing), knowledge discovery in water quality [30] and molecular biology databases [55] (data mining and discovery), design from first principles [8] (design and configuration), and finding functional dependencies [22, 34] (database design).

Acknowledgements

The research summarized in this paper was in part supported by the ESPRIT III Project 6020 Inductive Logic Programming. The authors acknowledge the financial support of the Slovenian Ministry of Science and Technology. Sašo Džeroski is currently also supported by the European Research Consortium for Informatics and Mathematics. Thanks to Luc De Raedt and Nada Lavrač for their useful comments and suggestions.

References


