Machine Learning, 41, 217–251, 2000

Phase Transitions in Relational Learning

ATTILIO GIORDANA attilio.giordana@unipmn.it
LORENZA SAITTA lorenza.saitta@unipmn.it
Dipartimento di Scienze e Tecnologie Avanzate, Università del Piemonte Orientale, Corso Borsalino 54, 15100, Alessandria, Italy

Editor: Raymond J. Mooney

Abstract. One of the major limitations of relational learning is due to the complexity of verifying hypotheses on examples. In this paper we investigate this task in light of recent published results, which show that many hard problems exhibit a narrow "phase transition" with respect to some order parameter, coupled with a large increase in computational complexity. First we show that matching a class of artificially generated Horn clauses on ground instances presents a typical phase transition in solvability with respect to both the number of literals in the clause and the number of constants occurring in the instance to match. Then, we demonstrate that phase transitions also appear in real-world learning problems, and that learners tend to generate inductive hypotheses lying exactly on the phase transition. On the other hand, an extensive experimenting revealed that not every matching problem inside the phase transition region is intractable. However, unfortunately, identifying those that are feasible cannot be done solely on the basis of the order parameters. To face this problem, we propose a method, based on a Monte Carlo algorithm, to estimate on-line the likelihood that the current matching problem will exceed a given amount of computational resources. The impact of the above findings on relational learning is discussed.

Keywords: complexity, phase transitions, relational learning, matching problem

1. Introduction

Recent investigations have revealed that several classes of computationally difficult problems, notably NP-complete problems, show a "phase transition" with respect to some typical order parameter, i.e., they present abrupt changes in their probability of being solvable, usually coupled with a peak in computational complexity (Cheeseman, Kanefsky, & Taylor, 1991; Williams & Hogg, 1994; Hogg, Huberman, & Williams, 1996b; Walsh, 1998). This phenomenon is typically true for search problems, and it seems to be largely independent from the specific search algorithm used (Hogg, Huberman, & Williams, 1996a). Phase transitions have been previously observed, for instance, in the K-Satisfiability problems (K-SAT) (Cheeseman, Kanefsky, & Taylor, 1991; Crawford & Auton, 1996; Freeman, 1996; Selman & Kirkpatrick, 1996), in Constraint Satisfaction problems (Smith & Dyer, 1996; Williams & Hogg, 1994; Prosser, 1996), in graph K-coloring problems (Cheeseman, Kanefsky, & Taylor, 1991; Hogg, 1996), and in the decision version of the Traveling Salesperson problems (Gent & Walsh, 1996; Zhang & Korf, 1996).

The detection and location of a phase transition in a problem class may have important consequences in practice. In fact, the standard notion of computational complexity of a class of problems is a pessimistic evaluation, based on worst-case analysis. Actually, many

problem instances can be solvable (or proved unsolvable) with reduced computational efforts. The investigation of phase transitions can provide information on single instances of the class, shifting the focus of the complexity analysis from the maximum complexity to a *typical* one. A phase transition divides the problem space into three regions: one, the NO-region, in which the probability that a solution exists is almost zero, and, hence, unsolvability is *easily* proved. Another one, the YES-region, where problems have many alternative solutions, and finding one is *easy*. Finally, a region where the probability of solution changes abruptly from almost 1 to almost 0, potentially making very difficult to find one of the existing solutions or to prove unsolvability. This region is called the *mushy* region (Smith & Dyer, 1996; Prosser, 1996).

In the present work we explore the emergence of a phase transition in the matching problem: possible models of a First Order Logic (FOL) formula are searched for in a given universe, in order to decide its satisfiability. Our basic motivation for studying the matching problem is the fundamental role it plays in learning structured concept descriptions from examples (Michalski, 1980; Bergadano, Giordana, & Saitta, 1988; Muggleton, 1992; Anglano et al., 1997, 1998; Giordana et al., 1997). The exponential (in time and/or space) complexity of this task severely limits the classes of concepts that can be learned and used. An effort to better understand the source of this complexity might suggest new and more effective heuristics or learning strategies.

Learning has been defined as a search problem in the space of possible *hypotheses* (Mitchell, 1982), and concept learning problems are known to be NP-hard even in a propositional setting (Hyafil & Rivest, 1976). The situation is worse in relational learning, because the complexity for searching the hypothesis space is combined with the complexity of verifying any single hypothesis against all positive and negative instances. Most relational learners adopt more or less strong biases, in order to limit such complexity, focusing on hypotheses that are easy to verify. We will propose here an alternative method to overcome this problem, allowing for weaker biases. Some relations between this approach and other approaches to taming complexity in relational learning will be discussed later.

The matching problem is firstly framed as a Constraint Satisfaction Problem (CSP) (Williams & Hogg, 1994), so that existing theories can be applied for interpreting the experimental results. A thorough experiment, involving the generation of thousands of problems according to a specified probability distribution, has been carried out. The chosen order parameters are different from (but related to) those usually adopted in generic CSPs. The reason is that we favored parameters with a more direct meaning in Machine Learning (ML). Also, the variability range of the parameters is close to the one occurring in ML practice.

Yet, one might legitimately wonder whether the results obtained for artificial classes of problems bear any relation to concrete learning problems. To answer this question we selected two real-world learning tasks for examination, and found, surprisingly as it might appear, that the results from the artificial cases still hold.

Having shown that the emergence of a phase transition may be relevant for learning, we try, as a second step, to predict where the mushy region might be, either to avoid it, or to design strategies better suited to limit the search complexity. A major finding of this analysis is that relational learners are *attracted* by the mushy region, making thus very

difficult avoiding it. At the same time, a large variability in complexity emerged inside this region, and, hence, many problems are tractable, in practice. In order to uncover which ones, we describe a stochastic algorithm for on-line estimation of the complexity of a single matching problem.

The body of this paper is organized as follows: Section 2 recalls previous theoretical analyses on Constraint Satisfaction Problems (Smith & Dyer, 1996; Williams & Hogg, 1994; Prosser, 1996). Section 3 links matching with CSP, and discusses the results of an extensive experiment supporting the emergence of a phase transition in matching. Section 4 presents two real-world case-studies, namely the analysis of an ML benchmark known as the *Mutagenesis* dataset (Srinivasan, Muggleton, & King, 1995), and of a troubleshooting problem (Giordana, Saitta, & Bergadano, 1993). In Section 5 the impact of the emergence of phase transitions on learning relations is discussed. Section 6 proposes a method for on-line identification of those instances of matching problems that are too hard to be handled, and Section 7 contains some conclusions.

2. Constraint satisfaction problems

Phase transitions have been widely investigated in the class of Constraint Satisfaction Problems (CSP) (Williams & Hogg, 1994; Smith & Dyer, 1996; Prosser, 1996). Given a set of variables $X = \{x_1, x_2, \ldots, x_n\}$, where each x_k ranges on a domain Λ_k of cardinality L_k, the problem of finding an assignment to each variable $x_k \in \mathbf{X}$ consistent with a set $\mathbf{R} = \{R_1, R_2, \ldots, R_m\}$ of constraints on $\mathbf{X}$, is called a CSP. A relation R, involving variables $(x_i, \ldots, x_j)$ can be represented as a table, where every row contains an admissible assignment $(a_i, \ldots, a_j)$ of constants to $(x_i, \ldots, x_j)$. If all relations are binary, the CSP is called binary (Williams & Hogg, 1994; Prosser, 1996; Smith & Dyer, 1996).

Two parameters are defined to characterize a CSP instance: *constraint density* p_1, and *constraint tightness* p_2 (Smith & Dyer, 1996; Prosser, 1996). In a binary CSPs the constraints can be represented as edges on a graph with n vertices, each one corresponding to a variable. The graph has $n(n-1)/2$ possible edges; several constraints on the same pair of variables can be reduced to a unique one by AND-ing them. Denoting by c the number of edges occurring in the constraint graph, the constraint density p_1 is defined (Smith & Dyer, 1996; Prosser, 1996) as follows:

$$p_1 = \frac{c}{\frac{n(n-1)}{2}} = \frac{2c}{n(n-1)} \tag{1}$$

The parameter p_1 belongs to the interval [0, 1], with 0 corresponding to no constraints at all, and 1 corresponding to the case in which all possible pairs of variables are constrained. The tightness of the constraint on a variable pair $\{x_i, x_j\}$ is the fraction of value pairs ruled out by the constraint itself. If N is the cardinality of the relation $R(x_i, x_j)$, the constraint tightness p_2 is defined (Smith & Dyer, 1996; Prosser, 1996) by:

$$p_2 = 1 - \frac{N}{L^2} \tag{2}$$

where L is the cardinality of the domain of the variables, assuming that every x_k ranges over the same set Λ. Studies on CSPs assume a model of stochastic instance generation; one of them, Model B (Smith & Dyer, 1996), assumes that n, N, and p_1 are kept constant, whereas p_2 varies in the interval [0, 1]. Edges on the constraint graph and tuples in the relations are extracted with uniform probability.

By varying p_2, a narrow mushy region, where the probability of solution P_{sol} drops from 0.99 to 0.01, is found (Williams & Hogg, 1994; Smith & Dyer, 1996; Prosser, 1996). The complexity of finding one solution (or of proving unsolvability) shows a marked peak in correspondence to $P_{\text{sol}} = 0.5$, which is also called the *crossover point* (Crawford & Auton, 1996; Smith & Dyer, 1996). Unsolvable instances require, on average, a greater complexity at the phase transition, because the whole search tree needs to be explored, and the tree is large. The p_2 value corresponding to the crossover point, $p_{2,\text{cr}}$, is called *critical* value; it is conjectured to correspond to an expected number of solutions roughly equal to 1 (Williams & Hogg, 1994; Smith & Dyer, 1996; Prosser, 1996; Gent & Walsh, 1996).

Even though the location and the height of the complexity peak depends upon the search algorithm used, the very emergence of the phase transition does not (Hogg, Huberman, & Williams, 1996a). For fixed n, N, p_1 and p_2, the actual location of $p_{2,\text{cr}}$ also depends upon the structure of the constraint graph. Williams and Hogg (1994), Prosser (1996), and Smith and Dyer (1996) all derive the same estimate, $\hat{p}_{2,\text{cr}}$, for the critical value of p_2:

$$\hat{p}_{2,\text{cr}} = 1 - L^{-\frac{2}{p_1(n-1)}} = 1 - L^{-\frac{n}{c}} \tag{3}$$

The estimate $\hat{p}_{2,\text{cr}}$ can be used to predict the location of the phase transition. Formula (3) has been derived by assuming that the average number of solutions at the phase transition is 1. However, the value $\hat{p}_{2,\text{cr}}$ given by (3) is a good predictor only for high values of p_1, or for large values of n. If p_1 is low, typically $p_1 < 0.3$, the constraint graph is sparse and many alternative configurations may exist, loosening the correspondence between $p_{2,\text{cr}}$ and the actual location of the phase transition. In this case, different constraint graphs correspond to different locations of the phase transition, and the mushy region derives from a mixture of different graphs, whose crossover points vary. When this happens, an average number of solutions equal to 1 may not denote an equal proportion of solvable and unsolvable problems (i.e., $P_{\text{sol}} = 0.5$), but rather the presence of a large number of unsolvable problems and a small number of solvable problems with many solutions; as a consequence, the $\hat{p}_{2,\text{cr}}$ value given by (3) is situated to the right of the actual phase transition. The reliability of $\hat{p}_{2,\text{cr}}$ as a predictor can be evaluated by estimating the expected number of solutions and its variance (Smith & Dyer, 1996).

In order to quantify the *constrainedness* of search, Gent and Walsh (1996) have proposed a different parameter:

$$\kappa = 1 - \frac{\log_2 E(N_{\text{sol}})}{\log_2 S}, \tag{4}$$

where $E(N_{\text{sol}})$ is the expected number of solutions existing in a search space with S states. Again assuming that the phase transition occurs for $E(N_{\text{sol}}) = 1$, the critical value of κ is

$\kappa_{\rm cr} = 1$. For a CSP of the type considered in this paper, formula (4) gives:

$$\kappa = -\frac{c \log_2(1 - p_2)}{n \log_2 L} \tag{5}$$

By setting $\kappa_{\rm cr} = 1$, the same expression (3) is obtained for the corresponding $\hat{p}_{2,\rm cr}$.

3. Phase transitions in matching

The matching problems we consider are restricted to the satisfiability of existentially quantified, conjunctive formulas, $\exists\vec{x}[\varphi(\vec{x})]$, with n variables (from a set **X**) and m literals (predicates from a set **P** or their negation). Given a universe U, consisting of a set of relations (tables) containing the extensions of the atomic predicates, the considered formula is satisfiable if there exists at least one model of $\varphi(\vec{x})$ in U. It is obvious that the matching problem is a CSP, where the n variables and their associated domains play the same role, and the m relations, corresponding to the literals occurring in $\varphi(\vec{x})$, correspond to the set **R** of relations. In learning relational concepts, a formula is a "hypothesis" (i.e., a putative description) and a universe is one positive or negative example of the concept to learn. Then, during learning, each hypothesis generated by the learner has to be matched to all the training examples, each one corresponding to a different universe. In relational learning, concept definitions are usually represented in DNF, i.e., as disjunctions of conjunctive formulas.

In order to investigate the location and properties of phase transitions in matching, formulas and examples have been generated according to a stochastic procedure. The following assumptions have been adopted:

- The variable $x_1, x_2, \ldots, x_n$ range over the same set Λ of constants, containing L elements.
- All the predicates are binary.
- Every relation in U has the same cardinality, namely it contains exactly N tuples (pairs of constants).

Given **X** and **P**, with the additional constraint $m \geq n - 1$, a formula φ with the structure below is generated, according to a random procedure described by Botta, Giordana, and Saitta (1999):

$$\varphi(\vec{x}) = \bigwedge_{i=1}^{n-1} \alpha_i(x_i, x_{i+1}) \wedge \bigwedge_{i=n}^{m} \alpha_i(y_i, z_i) \tag{6}$$

In (6), the variables $\{y_i, z_i\}$ belong to **X**, and $y_i \neq z_i$. The generated formulas contain exactly n variables and m literals, and the same pair of variables may appear in more than one predicate. The first part of formula (6) guarantees that the underlying constraint graph is connected, in order to hinder the matching problem from being reduced to simpler subproblems, with disjoint sets of variables.

Every relation in U is built up by creating the Cartesian product $\Lambda \times \Lambda$ of all possible pairs of constants, and selecting N pairs from it, uniformly and without replacement. In

this way, the same pair cannot occur twice in the same relation. This generation procedure is close to Model B for CSPs (Smith & Dyer, 1996).

In essence, a matching problem is defined by the 4-tuple (n, N, m, L), instead of the triple (n, p_1, p_2) usually employed in CSP. The parameters N, m and L can be rewritten in terms of p_1 and p_2, but these last do not have a direct meaning for learning problems. On the contrary, the complexity of an inductive hypothesis is frequently measured by m, and the complexity of a concept instance can be related to L, i.e., the number of atomic objects (ground literals) it contains. However, we will also use p_2 when the analysis requires it, as in Section 4.

3.1. Stochastic search algorithm

Given a formula φ with n variables and the syntactic structure (6), and given a universe U, the search for the models of φ in U entails visiting a tree τ. A node ν at level k in τ corresponds to a legal substitution θ for the variables $x_1, \ldots, x_k$, considered in a given sequence.[1] The leaves of τ at level $k = n$ represent models of φ, and are solutions to the matching problem.

Depending upon the strategy used for visiting τ, different algorithms show different search complexity. A comparison between a backtrack deterministic and a stochastic search algorithm has been presented by Botta, Giordana, and Saitta (1999). In the present paper we have used the stochastic one, because it offers two advantages for our purposes: On the one hand, it exhibits, in practice, an average complexity and a complexity variance lower than the deterministic one. Moreover, the algorithm is well suited to perform the on-line estimation of the search complexity that will be discussed in Section 6.

The search algorithm consists of the iteration of a one-step stochastic search function until either a model is found or the whole tree has been explored unsuccessfully. Let $\mathrm{MC}(\tau, n)$ be this function:

MC(τ, n)
 $\nu = \nu_0$, *leaf* = *False*
 while$(\neg$*leaf*$)$ **do**
  if ν *is a leaf at level k*
   then *leaf* = *True*
   else *Identify the Selectable children of* ν, *and put them into a set* $C(\nu)$
    Extract a node ν' *from* $C(\nu)$ *with uniform probability*
    Set $\nu = \nu'$
  endif
 end
 Label ν *as closed*
 if *the level of* ν *is* $k = n$ **then** *answer YES* **else** *answer NO*.

Function $\mathrm{MC}(\tau, n)$ implements a Monte Carlo algorithm (Brassard & Bratley, 1988), because it always provides an answer, but the answer may be incorrect; MC explores one path on the search tree, starting from the root ν_0 and ending in a leaf ν, which may or may not be a solution. The parameters τ and n of the function denote the search tree and the number

of variables (maximum depth of the tree), respectively. During the algorithm execution, ν is associated to a sequence of nodes in the tree at increasing depth, and corresponds to increasingly complete, legal partial assignments of values to the variables $x_1, \ldots, x_n$. By iterating MC on τ, more and more paths are explored.

Depending on the semantics of the criterion *Selectable*, different sets of child nodes of ν are included in $C(\nu)$. In the simplest case, all nodes are always *Selectable*, and the stochastic search is made with replacement: any leaf can be reached repeatedly. In this case the complete exploration of τ may asymptotically require an infinite number of repetitions of MC. If a search without replacement must be realized, the *Selectable* predicate shall not include in $C(\nu)$ any node that either is closed or has only closed children. In this case, every iteration of MC ends in a different leaf of τ and the whole tree is guaranteed to be completely explored with at most the same complexity as an exhaustive, backtrack search algorithm. The reported experiments have been done using the option of search without replacement.

In order to locate a possible phase transition, we have explored points in the (m, L) plane, for values of the number of variables $n = 4, 6, 10, 12, 14$, and cardinality of the relations in the universe $N = 50, 80, 100$ and 130. For each pair (n, N) the complete mesh, covering the region $(10 \leq L \leq 50, n - 1 \leq m \leq 50)$ in the plane (m, L), has been considered. For each pair (m, L) belonging the mesh, 100 problems have been generated for a total of about 900,000 problems. The range of n has been chosen consistently with the one employed in Machine Learning, where only a few variables have been considered so far.

3.2. Probability of solution

A 3-dimensional plot representing the probability of solution P_{sol} as a function of m and L is reported in figure 1, for $n = 10$ and $N = 100$. For each point in the mesh, P_{sol} has been computed as the fraction of problems with a solution among all the generated ones.

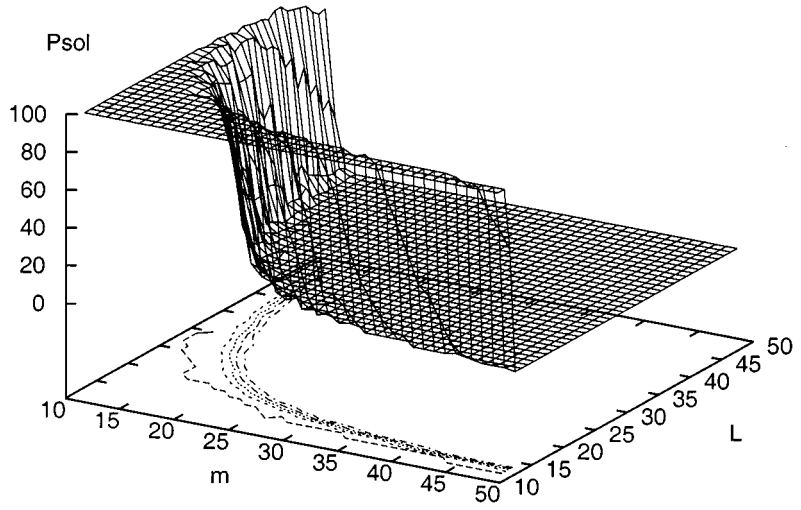


Figure 1. 3-Dimensional plot of the probability of solution P_{sol} for $n = 10$ and $N = 100$. Some contour level plots, corresponding to P_{sol} values in the range $[0.85 \div 0.15]$, have been projected onto the plane (m, L).

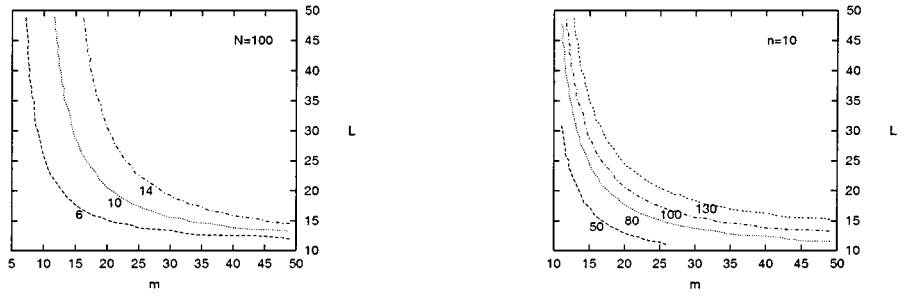


Figure 2. Plots of the 0.5-level contour of the probability of solution P_{sol}. (a) Graphs corresponding to numbers of variables $n = 6$, 10, and 14, with $N = 100$. (b) Graphs corresponding to relation cardinalities $N = 50$, 80, 100, and 130, with $n = 10$.

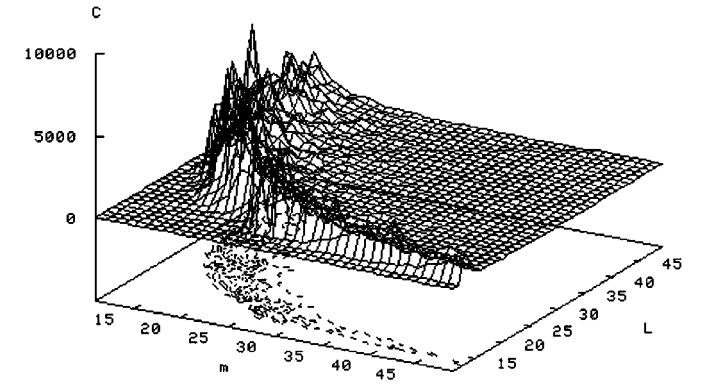
The graph in figure 1 has a noteworthy feature, namely its striking steepness. To the left of the steep descent (YES-region), all problems had a solution, whereas, to the right (NO-region) no solution could be found. Another interesting feature is the regularity of the projection, onto the (m, L) plane, of the contour level plot at $P_{\text{sol}} = 0.5$, which is a very smooth curve with a hyperbolic-like behavior. Figure 2(a) reports the projections of the contour level plots at $P_{\text{sol}} = 0.5$ for numbers of variables $n = 6$, 10 and 14. Figure 2(b) reports an analogous set of contour plots for a constant number of variables $n = 10$, and for cardinalities of the relations $N = 50$, 80, 100 and 130.

3.3. Search complexity

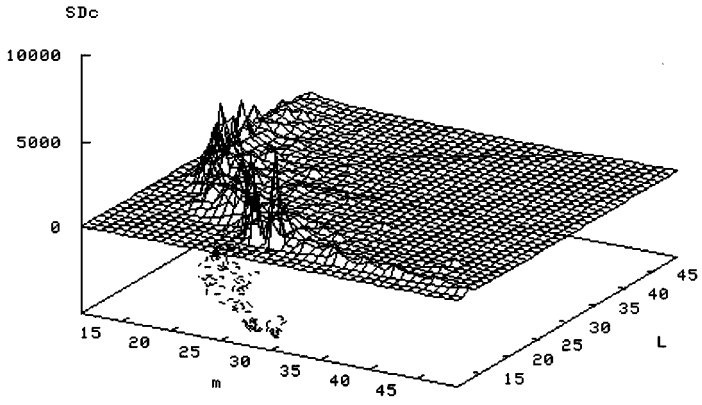
For a quantitative analysis of the complexity, a random search without replacement was performed by repeating the Monte Carlo algorithm described in Section 3.1. The cost C of the search has been defined as the total number of explored nodes in the search tree, until either a first solution is found, or unsatisfiability is proved. For unsatisfiable problems it is necessary to explore the whole tree.

In figure 3(a), the graph of the search complexity C, averaged over the 100 repetitions for each point, is reported, for $n = 10$ and $N = 100$. The shape and location of the highest complexity region roughly matches the transition in probability reported in figure 1, but it is more irregular and also broader, like a "mountain chain". Inside the "mountain", there is a large variability among different problems, witnessed by the variance plot, reported in figure 3(b). As one may expect, the highest variance values correspond to the highest peaks. The maximum complexity contour coincides with the contour plot at $P_{\text{sol}} = 0.5$, as it has been found previously (Hogg, Huberman, & Williams, 1996a; Hogg, 1996).

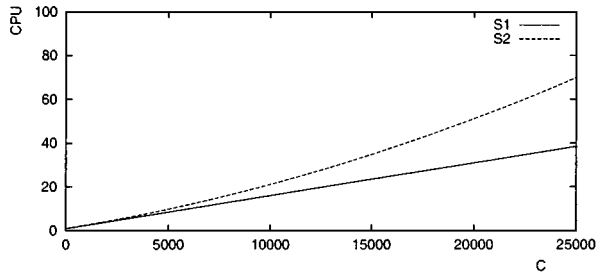
It is worth noticing that the complexity distributions for solvable and unsolvable problems may be very different, as the unsolvable problems usually require much more search. Approximations to the complexity probability distributions at the phase transition for solvable and unsolvable CSPs are provided by Frost, Rish, and Vila (1997). They show that



(a)



(b)



(c)

Figure 3. (a) Plot of the complexity C of the Monte Carlo stochastic search algorithm MC without replacement, for $n = 10$ and $N = 100$. Each point is the average over 100 problem instances. (b) Plot of the standard deviation of the complexity. (c) CPU time, for a single matching, in centi-seconds versus the complexity of the search for two different implementations, S_1 and S_2, of the stochastic matching algorithm. The curves have been obtained by interpolating the measures obtained for each one of the runs reported in figure 3(a) and (b).

a LogNormal distribution is a good approximation for unsolvable problems. For solvable problems several known distributions (in particular, a Weibull distribution) have been tried with less success. However, from their reported experiments it clearly emerges that the complexity distribution of both solvable and unsolvable problems has a long tail in the region of extremely hard problem instances.

Finally, figure 3(c) shows the dependency of the CPU time upon the number of explored nodes, for two different implementations of the stochastic matching algorithm: S_1 and S_2. Implementation S_1 stores the entire search tree so that the time complexity is linear in the number of nodes, but the memory requirement is very heavy. On the contrary, implementation S_2 makes use of implicit indexing in order to avoid storing the pointers from a node to its children. Then the memory request is more modest but there is an extra cost in CPU time, which is quadratic in C. Implementation S_2 is a reasonable trade-off, and has been used for most experiments reported here.[2] The measures have been obtained using a Pentium II-366.

4. Two real-world case studies

Up to now we have been concerned with an ensemble of randomly generated matching problems. One may wonder whether phase transitions do occur in real life, and whether they have an impact on real-world learning problems.

Other authors have already shown that phase transitions do emerge in real-world problems that cannot be supposed randomly generated. For instance, Gent and Walsh (1996) have analyzed the Travelling Salesperson problem on a city graph containing the capitals of 48 contiguous states of the USA. A phase transition did occur, although at a smaller control parameter value than for random graphs, whereas the cost of search was higher than predicted. The same authors have also noticed a phase transition in graph coloring problems derived from university exam time-tables (Gent & Walsh, 1995), whereas Gomes and Selman (1997) found a phase transition in quasi-group completion.

Given a real-world problem, in order to interpret the emergence of an ensemble phenomenon like a phase transition, one has to hypothesize that the problem is extracted from a population of problems having the same values of the order parameters as the ones considered. Learning is an anomalous task, in this respect. In fact, the ensemble of problems to consider for the emergence of phase transitions is generated *internally* by the learner itself. Actually, the set of training examples is given, but the learner generates many candidate hypotheses during search. Each example is paired with each hypothesis, generating thus a possibly large number of matching problems. Given a specific learning task, including a set of training examples, learners differ among each other for the way in which they generate hypotheses, i.e., for the heuristics they use. Different heuristics might correspond to phase transitions of different location and steepness, and the ensemble of matching problems they give birth to may be more or less similar to the randomly generated set.

In this section we analyze two real-world learning problems using G-Net, a relational learner based on an evolutionary search strategy guided by the Minimum Description Length (MDL) (Anglano et al., 1997, 1998). The datasets suitable for relational learning, available

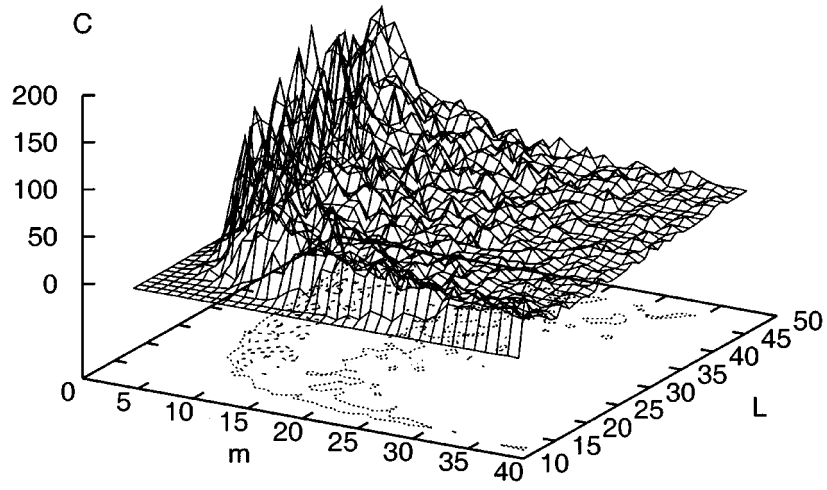


Figure 4. Complexity in the (m, L) plane for randomly generated matching problems with $n = 4$ and $N = 100$.

in public repositories, are few and, in general, rather simple. In fact, the concept descriptions that have been learned from them contain few literals and, mostly, two or three chained variables. The datasets we selected are among the most complex we found, as for both of them descriptions containing up to 4 variables and up to 6 binary relations have been discovered. For the sake of reference, figure 4 reports the same graph as figure 3(a), but for $n = 4$. A phase transition is evident, but the expected complexity in the mushy region is much lower.

Comparing figure 4 ($n = 4$) with figure 3(a) ($n = 10$), we notice that the mushy region is much wider for $n = 4$ than for $n = 10$, as predicted by the theory (Williams & Hogg, 1994). Moreover, a 50-fold increase in the complexity is observed in correspondence to a 2.5 increase in the number of variables.

4.1. Mutagenesis dataset

In this subsection we consider a learning problem known as the *Mutagenesis* in the Machine Learning community, where it is used as a benchmark for testing induction algorithm in First Order Logic: the prediction of mutagenicity in nitroaromatic chemical compounds on the basis of their structure (Srinivasan, Muggleton, & King, 1995). The goal of our analysis is to investigate where the classification rules learned by an inductive program lay in the plane (m, L), with respect to the mushy region.

The Mutagenesis dataset[3] consists of the chemical description of 188 molecules, classified as "mutagenic" (125 positive examples) or "non mutagenic" (63 negative examples). The goal of the learning task is to discover classification rules that separate the two classes. Every compound is described as a set of atoms, each one characterized by an attribute vector reporting the atom type, the atomic number, and the electrical charge, plus a set

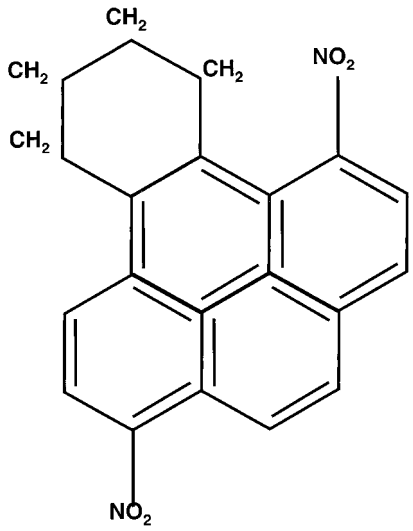


Figure 5. Example of a nitroaromatic molecule's structure, in the Mutagenesis dataset. Each atom is denoted by a constant, and each link defines a binary relation between two atoms.

of relations describing atomic links and substructures of the molecule, such as aromatic rings and others. Moreover, every compound is characterized by two global numeric attributes: *lumo* and *logp*, corresponding to the energy of the compound's lowest unoccupied molecular orbital, and the logarithm of the compound's octanol/water partition coefficient, respectively. An extensive experimentation with different sets of attributes is reported by Srinivasan, Muggleton, and King (1995).

The definition of this learning problem is usually based upon predicates (constraints) with arity greater than 2, and it is not immediately suitable for being analyzed with the method of Section 3, limited to binary constraints.[4] However, the problem can be reformulated using only unary and binary predicates, as done by Anglano et al. (1998). Every molecule is considered as a different universe that must be classified as either mutagenic or not. The hypothesis description language contains literals of the form $P(x, K)$ or $Q(x, y)$, where variable x and y range on atoms, and K denotes a set of constants, which are to be learned by the induction algorithm (Giordana et al., 1997). In figure 5 an example of molecule is reported.

Two sets of experiments have been performed with two different hypothesis description languages, $\mathcal{L}_1$ and $\mathcal{L}_2$. The language $\mathcal{L}_1$ is analogous to the one used by other authors in the past (Sebag & Rouveirol, 1997, 2000), and contains three unary predicates, namely, $chrg(x, K)$, reporting the electrical charge, $anm(x, K)$, reporting the atomic number, and $type(x, K)$, reporting the atomic type, plus one binary predicate, $bound(x, y)$, stating the existence of a link between two atoms. Moreover, the constraint $x < y$ has been imposed for every variable pair in order to avoid inefficiency, due to the test of symmetric or reflexive relations entailed by the relation $bound(x, y)$. The language $\mathcal{L}_2$ contains all the predicates defined in $\mathcal{L}_1$ with the addition of $lumo(x, K)$ and $logp(x, K)$ to the description of each

$$\varphi_1 : \; anm(x_3,[195,22,3,27,38,40,92]) \wedge \neg chrg(x_3,[-0.2,0.2]) \wedge$$
$$anm(x_4,[195,22,3,38,40,29,92]) \wedge \neg type(x_4,[O]) \wedge \neg chrg(x_4,[-0.2]) \wedge$$
$$(x_1 < x_2) \wedge (x_1 < x_3) \wedge (x_1 < x_4) \wedge (x_2 < x_3) \wedge (x_2 < x_4) \wedge (x_3 < x_4) \wedge$$
$$bound(x_3,x_4) \;\rightarrow\; mutagenic$$

$$\varphi_2 : \; \neg chrg(x_1,[-0.2]) \wedge \neg type(x_2,[N]) \wedge \neg anm(x_3,[22]) \wedge \neg chrg(x_3,[-0.6,-0.4]) \wedge$$
$$\neg type(x_4,[H,N,O]) \wedge (x_1 < x_2) \wedge (x_1 < x_3) \wedge (x_1 < x_4) \wedge (x_2 < x_3) \wedge$$
$$bound(x_2,x_3) \wedge (x_2 < x_4) \wedge (x_3 < x_4) \wedge bound(x_3,x_4) \;\rightarrow\; mutagenic$$

$$\varphi_3 : \; anm(x_1,[195,38,29,92]) \wedge chrg(x_1,[-0.8 \div 0.6]) \wedge \neg type(x_3,[C]) \wedge \neg chrg(x_3,[0.0]) \wedge$$
$$anm(x_4,[195,22,3,27,38,29,92]) \wedge \neg type(x_4,[N]) \wedge (x_1 < x_2) \wedge (x_1 < x_3) \wedge$$
$$(x_1 < x_4) \wedge (x_2 < x_3) \wedge (x_2 < x_4) \wedge (x_3 < x_4) \;\rightarrow\; mutagenic$$

$$\varphi_4 : \; anm(x\ 1,[195,3,27,38,40,29,92]) \wedge \neg type(x_1,[H]) \wedge \neg chrg(x_1,[-0.2])$$
$$\neg anm(x_3,[40]) \wedge anm(x_4,[195,22,27,38,40,29,92]) \wedge \neg type(x_4,[H,N])$$
$$(x_1 < x_2) \wedge \neg bound(x_1,x2) \wedge (x_1 < x_3) \wedge (x_1 < x_4) \wedge (x_2 < x_3) \wedge (x_2 < x_4) \wedge$$
$$bound(x_{,}x_{)} \wedge (x3 < x4) \;\rightarrow\; mutagenic$$

Figure 6. Solution Φ, learned by G-Net using the language $\mathcal{L}_1$. Φ correctly classifies 94.1% of the whole dataset.

atom. G-Net was forced to generate formulas with exactly four variables, which is the maximum number used in previous studies. In both experiments, G-Net was run several times on the entire dataset of 188 examples, producing sets of classification rules correctly covering from 90% to 95% of the examples, depending on the control parameter setting.[5]

In the following we will analyze in detail two solutions, namely $\Phi = \{\varphi_1, \varphi_2, \varphi_3, \varphi_4\}$, consisting of the four clauses reported in figure 6, expressed in the language $\mathcal{L}_1$, and $\Psi = \{\psi_1, \psi_2, \psi_3\}$, consisting of three clauses reported in figure 7, expressed in language $\mathcal{L}_2$. The same analysis has been performed on several other solutions generated by G-Net, obtaining qualitatively equivalent results.

$$\psi_1 : \; first-atom(x_1) \wedge logp(x_1,[0.0 \div 7.0]) \wedge \neg lumo(x_1,[-1.0]) \wedge \neg logp(x_2,[1.5,7.0]) \wedge$$
$$\neg lumo(x_2,[-1.25]) \wedge \neg logp(x_3,[0.5,1.0,6.5]) \wedge \neg lumo(x_3,[-4.0 \div -1.0]) \wedge$$
$$\neg logp(x_4,[2.5,3.0]) \wedge (x_1 < x_2) \wedge (x_1 < x_3) \wedge (x_1 < x_4) \wedge (x_2 < x_3) \wedge$$
$$(x_2 < x_4) \wedge (x_3 < x_4) \;\rightarrow\; mutagenic$$

$$\psi_2 : \; first-atom(x_1) \wedge logp(x_1,[0.0 \div 7.0]) \wedge \neg lumo(x_1,[-1.0]) \wedge \neg logp(x_2,[1.5]) \wedge$$
$$\neg lumo(x_2,[-1.25]) \wedge \neg logp(x_3,[0.5]) \wedge lumo(x_3,[-1.5,-0.75]) \wedge \neg logp(x_4,[2.5]) \wedge$$
$$\neg lumo(x_4,[-1.75]) \wedge (x_1 < x_2) \wedge (x_1 < x_3) \wedge (x_1 < x_4) \wedge (x_2 < x_3) \wedge$$
$$(x_2 < x_4) \wedge (x_3 < x_4) \;\rightarrow\; mutagenic$$

$$\psi_3 : \; first-atom(x_1) \wedge \neg lumo(x_1,[-1.0]) \wedge \neg logp(x_2,2.0]) \wedge$$
$$anm(x_3,[195,22,3,27,38,40,29,92]) \wedge \neg chrg(x_3,[-0.20)]) \wedge \neg anm(x_4,[22]) \wedge$$
$$type(x_4,[C,O,F]) \wedge \neg chrg(x_4,[-0.4,0.0]) \wedge (x_1 < x_2) \wedge (x_1 < x_3) \wedge (x_1 < x_4) \wedge$$
$$(x_2 < x_3) \wedge (x_2 < x_4) \wedge (x_3 < x_4) \;\rightarrow\; mutagenic$$

Figure 7. Solution Ψ learned by G-Net using the language $\mathcal{L}_2$. Ψ correctly classifies 90.7% of the dataset.

All rules in the solutions Φ and Ψ have been analyzed according to the following procedure: For each rule $\varphi_i \in \Phi$ or $\psi_i \in \Psi$, the two parameters p_2 and $\hat{p}_{2,\mathrm{cr}}$ have been computed for every example in the dataset, using expressions (2) and (3), respectively. The reason for using p_2 is twofold: on the one hand, m and n are constant for each formula, whereas L and N change from one example to another; this variability is captured by p_2, which depends upon both N and L. Then, theoretical results from the literature (Prosser, 1996) can be used directly.

For our analysis, every formula has been decomposed into subformulas with the following structure:

$$\gamma(x_1, x_2) = \alpha_1(x_1) \wedge \alpha_2(x_2) \wedge \beta(x_1, x_2) \tag{7}$$

Each subformula γ has been considered as a single constraint. The unary predicates occur in each subformula containing as argument the same variable; they have the role of reducing the number of bindings that may occur in the binary relations (namely, the N value). As all variables in a clause are correlated at least through the predicate "$<$", six binary formulas have always been obtained. Then, $p_1 = 1$ for every clause, whereas the parameter $\hat{p}_{2,\mathrm{cr}}$ depends upon the number L of constants. L corresponds, in this case, to the number of atoms in a molecule, and varies from one example to another. More precisely, the minimum value for L in the dataset is $L_{\min} = 18$, the maximum $L_{\mathrm{Max}} = 40$ and the average $L_{\mathrm{avg}} = 26.7$.

Using expression (3), we obtain, for all the considered formulas:

$$\hat{p}_{2,\mathrm{cr}} = 1 - L^{-\frac{2}{3}} \tag{8}$$

The parameter p_2 also depends upon the formula φ and upon the example corresponding to a universe U; in order to stress this dependency, we use the notation $p_2(\varphi, U)$. More specifically, p_2 has been computed according to the expression:

$$p_2(\varphi, U) = \frac{1}{6}\sum_{j=1}^{6} p_2(\gamma_j, U) = 1 - \frac{6}{L^2}\sum_{j=1}^{6} N_j \tag{9}$$

In (9) γ_j is one of the binary subformulas obtained from φ; its associated relation has N_j elements. Let us consider now the classification rules $\Phi = \{\varphi_1, \varphi_2, \varphi_3, \varphi_4\}$. For each rule φ_i we computed the distribution of the variable $(p_2 - \hat{p}_{2,\mathrm{cr}})$ over all the examples in the dataset, over the positive examples, and over the examples (both positive and negative) "covered" by the rule. The graphs of these distributions are reported in figure 8. If the matching problem corresponding to a <formula, example> pair is exactly on the phase transition, the value $(p_2 - \hat{p}_{2,\mathrm{cr}})$ should be zero. Notice that the mushy region is quite large for $n = 4$, as we can see from figure 4; moreover, as neither L nor N are constant across relations and examples, the broadening of the mushy region is enhanced. Figure 8 clearly shows that, for formulas φ_2, φ_3 and φ_4, the p_2 values are distributed substantially in the mushy region for both positive and negative examples, whereas the matching problems involving φ_1 seem to lay mostly in the YES-region.

The same analysis has been performed on solution Ψ, and the results are reported in figure 9. Solution Ψ shows a different behavior. In fact, rules ψ_1 and ψ_2 exhibit three separate

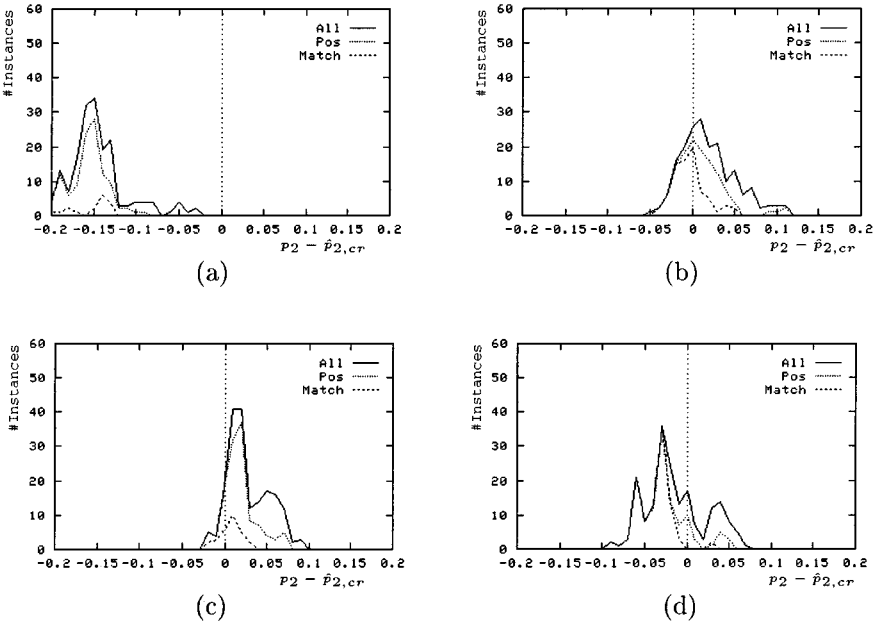


Figure 8. Distributions of the variable $(p_2 - \hat{p}_{2,\mathrm{cr}})$, reported on the x axis, for the Mutagenesis dataset and the formulas φ_1 in (a), φ_2 in (b), φ_3 in (c), φ_4 in (d). The y axis reports the number of examples (all, positive ones, and those covered by the formula) corresponding to a given value of $(p_2 - \hat{p}_{2,\mathrm{cr}})$.

peaks: one to the left, one inside, and one to the right of the mushy region, respectively. Moreover, the peaks corresponding to the examples satisfying the clause practically coincide with the left peak. A different behavior is exhibited by clause ψ_3, which shows only two peaks, the first one near the critical point $\hat{p}_{2,\mathrm{cr}}$, and the second one clearly to the right of the mushy region. This situation is confirmed by the presence of both positive and negative instances in the peak.

From figures 8(a)–(d) we would predict that formula φ_1 should be easy to match for all the examples, whereas φ_2 is likely to require a high computational cost to be matched, because most examples lay in the critical region. For formulas φ_3 and φ_4, many examples are close to the mushy region, but not exactly at the transition point, so that an intermediate complexity should be expected. In Table 1 the measured complexity for matching the formulas on the whole dataset is reported.

As we can see, the theory prediction for all the formulas is substantially verified, except for φ_1, for which both the location of the peak in figure 8(a) and the complexity in Table 1 appear to be wrong. By looking more closely at formula φ_1 in figure 6, we suggest the following explanation. Formula φ_1 actually contains only two "meaningful" variables, namely x_3 and x_4; then, $n = 2$ and $c = 1$. In this case, the estimated value $\hat{p}_{2,\mathrm{cr}}$ is actually a little larger than the one used in the figure. On the other hand, N is computed as average of all the relations involved in the formula, so that the extension of "$x_1 < x_2$", which is much larger than the other ones, lets p_2 appear much smaller than it must be. The consequence

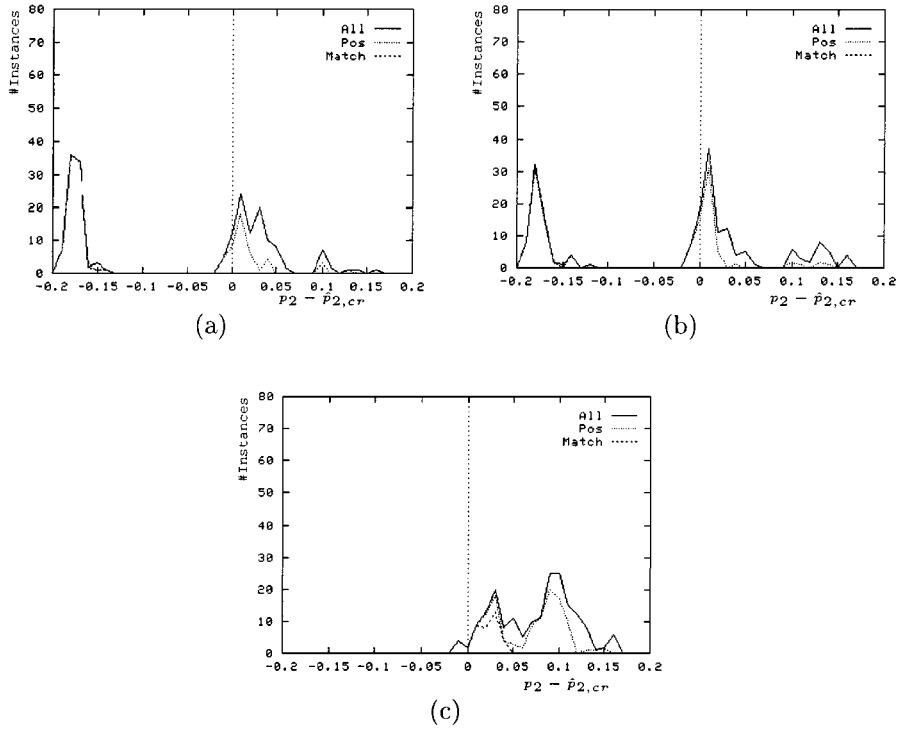


Figure 9. Distributions of the variable $(p_2 - \hat{p}_{2,\mathrm{cr}})$, reported on the x axis, for the Mutagenesis dataset and the formulas ψ_1 in (a), ψ_2 in (b), and ψ_3 in (c). The y axis reports the number of instances (all, positive ones, and those covered by the formula) for each value of $(p_2 - \hat{p}_{2,\mathrm{cr}})$.

is the apparent shift toward the left with respect to the phase transition. The second aspect to be explained, namely the abnormally high complexity in Table 1, is also related to the spurious joins of the intermediate tables corresponding to x_1 and x_2, which are pruned only later. This effect would not have appeared by exploiting a dynamic variable ordering during match. A set of focused experiments on φ_1, reduced to the subformula containing only x_3 and x_4, has confirmed both explanations. Among the seven formulas in Φ and Ψ, φ_1 is the only one in which only two variables are effective. It is sufficient that three among the

Table 1. Average complexity for matching the clauses in Φ and Ψ to all the examples in the dataset.

	Φ				Ψ		
	φ_1	φ_2	φ_3	φ_4	ψ_1	ψ_2	ψ_3
Avg	26215.10	5168.06	1249.04	1496.85	1.33	1.43	7.06
$\mathrm{Avg}_{\mathrm{pos}}$	22.46	207.74	23.89	1249.86	2.00	2.00	2.35
$\mathrm{Avg}_{\mathrm{neg}}$	30418.86	8609.00	1463.44	1789.79	1.00	1.00	8.33

Table 2. Classification rates obtained by setting a threshold between the peaks corresponding to low and high p_2 values, respectively, for the three formulae ψ_1, ψ_2, and ψ_3. The values between brackets correspond to the classification obtained by actually matching the formula on the dataset. Setting a threshold on p_2 reduces the omission error, but increases the commission error.

Formula	Threshold on p_2	Positive	Negative
ψ_1	0.85	80 (80)	3 (1)
ψ_2	0.85	60 (60)	4 (2)
ψ_3	0.95	54 (40)	23 (0)

four variables are chained by the predicate *bound*, which is much more constraining than predicate "$<$", to let the anomaly disappear.

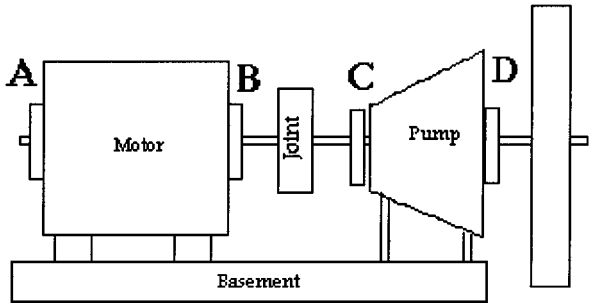
An interesting observation can be done on figure 9(a)–(c): The positive and negative examples could be discriminated almost without performing the matching, but only by setting a threshold on p_2; by considering "positive" the examples on the left and "negative" those on the right of the threshold, the classification reported in Table 2 is obtained. The values of p_2, and, hence, the threshold, can be computed from N and L only. Problems that exhibit this kind of behavior are essentially "propositional", even though formally expressed in a FOL language. The very low matching complexity in Table 1 confirms this assertion. The above property can be exploited to reduce the amount of matching to be done during learning and knowledge use. In fact, by estimating the distributions of p_2 values for the positive and negative training examples, a "best" threshold (or, better, a "best margin") can be learned.

Moreover, by looking at the syntactic structure of the clauses reported in Ψ, (see figure 7), we notice that most literals occurring in them deal with the attributes *lumo* and *logp*, which have the same value for all atoms, according to the way they have been defined. Therefore, in spite of its structural aspect, ψ_1 and ψ_2 are easily translated into propositional assertions. Rule ψ_3 shows a different structure, which contains also literals related to the atomic charge and the atomic number. This is sufficient to require an actual matching. This last situation occurs in all clauses of solution Φ.

4.2. Mechanical troubleshooting datasets

The second real-world case study is a problem that we approached some time ago in an industrial environment. The goal of the application was the automatic acquisition of a diagnostic knowledge base for mechanical troubleshooting at the chemical company ENICHEM, in Ravenna (Italy). The knowledge base learned by the system ENIGMA (Giordana et al., 1993) has been used for years by the company.

The basis for the troubleshooting was Mechanalysis, a methodology that exploits mechanical vibrations, and requires a strong expertise to be applied. The diagnosed apparati, ranging from small motor-pumps to very large turbo-alternators, shared the common feature of possessing a rotating shaft. When some fault occurs in the machine, anomalous vibrations appear. Mechanalysis basically performs a Fourier analysis of the vibratory motions



(a)

Support	Direction	Total Vibration		Fourier Analysis				
		Amplitude [μm]	Speed [mm/s]	ϖ [CPM]	v [mm/s]	.....	ϖ [CPM]	v [mm/s]
A	Hor	[7–11]	[2.4–2.6]	3000	[0.7–0.9]	.....	18,000	0.7
	Vert	[4–8]	[1.2–1.4]	3000	[0.2–0.7]	.....	18,000	0.4
	Ax	20	12	3000	[3-3.2]	.....	18,000	[0.8-1]
.....		.....		.....		.....	.....	

(b)

Figure 10. Structure of a mechanalysis table, corresponding to a single example. (a) Scheme of a motor-pump. The vibrations on the four supports A, B, C and D are measured. (b) For each support (A, B, C and D) and for each triple of "Total Vibration" measurements, several groups of three rows, such as the ones reported under "Fourier Analysis", may be present, as vibrations with different frequencies are measured. Globally, a mechanalysis table may contain 20 through 60 items, an item being an entry in the mechanalysis table, i.e., a 4-tuple <support, direction, frequency, velocity> for each vibration harmonic.

measured on the supports of the machine components. Each mechanalysis is an example. The data, arranged into groups, correspond to the machine's supports: Each group contains the measures of frequency and velocity of the harmonic components of the vibration for three spatial directions, as shown in figure 10.

The troubleshooting task consists in discriminating among six classes (one "normal" and five types of fault). G-Net found 13 conjunctive formulas distributed over the six classes,[6] each one with at most four variables. One of these formulas is the following:

$$\varphi = vout(x_1) \wedge sup(x_1, [2, 3, 4]) \wedge ismax(x_2) \wedge \neg mis(x_2, [0.0 - 3.0]) \wedge$$
$$vin(x_3) \wedge rpm(x_2, [2, 3, 4, 6, 7, 8]) \wedge \neg cpm(x_3, [9.0]) \wedge \neg mis(x_3, [1.0 \div 2.0]) \wedge$$
$$\neg fea(x_3, [ia, iv]) \wedge \neg rpm(x_3, [5]) \wedge \neg sup(x_4, [1, 3]) \wedge near(x_1, x_2, [1]) \wedge$$
$$near(x_1, x_3, [1]) \wedge \neg near(x_1, x_4, [1]) \wedge near(x_2, x_3, [0, 1])$$

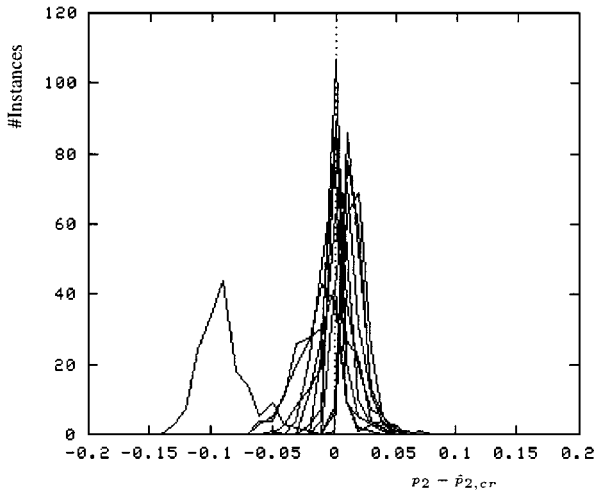


Figure 11. Distribution of the variable $(p_2 - \hat{p}_{2,\mathrm{cr}})$ for the matching problems obtained by pairing each of the 13 formulas (disjuncts) in the solution with all the examples in the dataset. Each graph corresponds to one of the 13 formulas.

The meaning of the predicates in φ is not important here, and can be found in previous works (Giordana et al., 1993). The relevant aspect, in this paper, is the syntactic structure of φ. In figures 11 and 12 the results of the same analysis that was performed on the Mutagenesis dataset are reported. More specifically, figure 11 reports the distribution of the variable $(p_2 - \hat{p}_{2,\mathrm{cr}})$ for the matching problems obtained by pairing each of the 13 formulas with all the examples in the dataset (164 examples), for a total of 2132 matching problems. In figure 12, on the contrary, only matching problems obtained by pairing each formula with the positive examples of its class are considered.

As we can see from figure 11, most problems lay inside the mushy region, except for one of the formulas. A closer analysis of this formula showed that, contrarily to the case of figure 4(a), the peak to the left of the phase transition actually corresponds to an "easy" problem, with a low matching complexity and a high coverage of both positive and negative examples.

In the two real-world problems we considered, the cardinality N of the relations corresponding to the basic predicates was not constant, as the random generation model assumes. Then, we have considered the model prediction for a range of N values corresponding to the actual cardinalities occurring in the two datasets. The plot in figure 13 is analogous to the one reported in figure 2(b), but for $n = 4$ variables. Again, N has been set to 50, 80, 100 and 130, respectively.

In figure 13, we have located the "average" solutions found by G-Net (averaged over all pairs <learned clause, example>), in the plane (m, L). As it appears from the figure, these solutions are located on the respective phase transition curves.

5. Relational learners work in the mushy region

The experiments with real datasets support the claim that phase transitions are relevant to relational learning. In fact, most concept definitions acquired by G-Net have been found

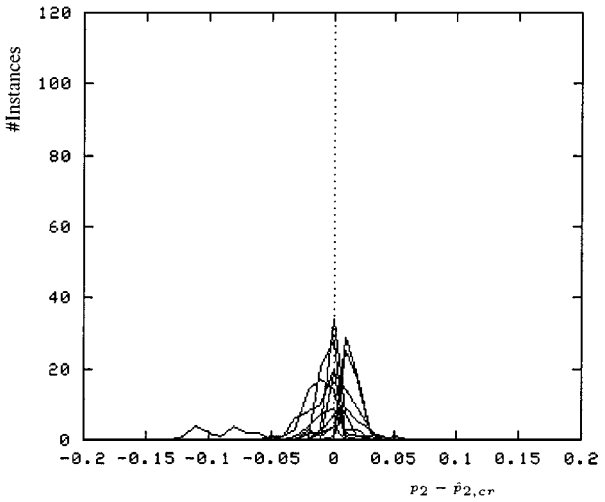


Figure 12. Distribution of the variable $(p_2 - \hat{p}_{2,\mathrm{cr}})$ obtained by matching each disjunct corresponding to a given class with the positive examples of the same class, covered by it. Hence, all the considered problems are solvable.

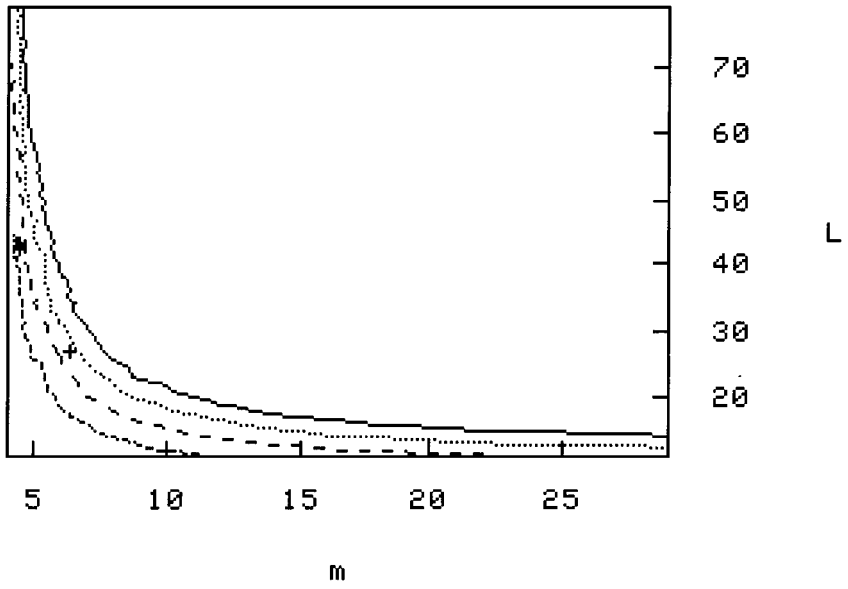


Figure 13. Location of the line $P_{\mathrm{sol}} = 0.5$ for $N = 50$, 80, 100, 130, and $n = 4$ variables. The symbols "+" and "*" locate the positions in the plane (m, L) of the "average" matching problem found in the Mutagenesis and Mechanical Troubleshooting datasets, respectively.

in the high complexity region of the (m, L) plane. Then, the inductive search must have occurred mostly in this region. Similar results have been presented by Botta, Giordana, and Saitta (1999), who have shown, using a large set of artificial problems, that also FOIL (Quinlan, 1990) systematically tends to generate concept descriptions located in the mushy region. In this section we will discuss this finding and its implication for learning.

As shown in the previous sections, matching problems in the NO-region are almost always unsolvable, but occasionally some of them are solvable. On the contrary, matching problems in the YES-region are usually solvable, but exceptionally some are unsolvable. In both NO- and YES-regions the matching complexity is usually low.

Let us now consider two examples of a concept ω, $\mathbf{e}_p$ and $\mathbf{e}_n$, one positive and one negative. Let L_0 be the average number of constants occurring in the two examples. We want to learn a concept definition ψ that covers $\mathbf{e}_p$ and does not cover $\mathbf{e}_n$. Given a concept description language L, the hypothesis space defined by L generates a set of matching problems corresponding to points on the horizontal line $L = L_0$ in the plane (m, L). This line intersects the mushy region. The results from the random problem generation tell us that any hypothesis for ω defining a matching problem in the NO-region has very little chance of being verified by $\mathbf{e}_p$ and $\mathbf{e}_n$. Then, it would be easy to exclude $\mathbf{e}_n$, but finding a definition for ω that covers $\mathbf{e}_p$ may turn out to be a very hard search problem, indeed. On the contrary, hypotheses generating matching problems in the YES-region tend to verify both $\mathbf{e}_p$ and $\mathbf{e}_n$. Then, it is easy to cover $\mathbf{e}_p$ but difficult to exclude $\mathbf{e}_n$. On the other hand, a hypothesis defining a matching problem on the phase transition has about 50% chance of verifying any instance, so that it should be easier to discriminate between $\mathbf{e}_p$ and $\mathbf{e}_n$.

In order to test the above conjecture, we have built up two instances, $\mathbf{e}_1$ and $\mathbf{e}_2$, each one with $L = 16$ constants. Moreover, 50 binary predicates have been defined, corresponding to relations containing $N = 100$ tuples. Finally, hypotheses with $n = 4$ variables have been created according to the procedure used in Section 3. More precisely, for each value of $m \in [3, 45]$, a thousand formulas have been generated, and 86,000 matching problems have been defined by pairing each formula with both $\mathbf{e}_1$ and $\mathbf{e}_2$. For each m value, the proportion of formulas, P_d, covering exactly one among $\mathbf{e}_1$ and $\mathbf{e}_2$ (discriminant formulas) has been computed, and reported in figure 14. For the sake of reference, also the graph of the probability of solution P_{sol} is reported.

From the graph, it clearly appears that the proportion of discriminant formulas is at its maximum when $P_{\text{sol}} = 0.5$, at the phase transition. Therefore, independently of the specific distribution of the concept instances, that portion of the hypothesis space that defines matching problems inside the mushy region has a much higher density of acceptable concept definitions than the other ones. In conclusion, we formulate the conjecture that any data-driven induction algorithm will most likely search in this region. The described behavior is reinforced by a search heuristic biased toward simplicity; in fact, a learner guided by such a heuristic will tend to focus the search where the hypotheses are discriminant and, at the same time, as simple as possible, i.e., in the mushy region. An extensive experimentation performed with FOIL (Quinlan, 1990) confirms the conjecture (Botta et al., 1999; Giordana et al., 2000).

To further test the above conjecture, we have analyzed the time evolution of the composition of the hypothesis population manipulated by the evolutionary learner G-Net, used for the case-studies reported in Section 4. Given a set of examples, figure 15(a) shows the distri-

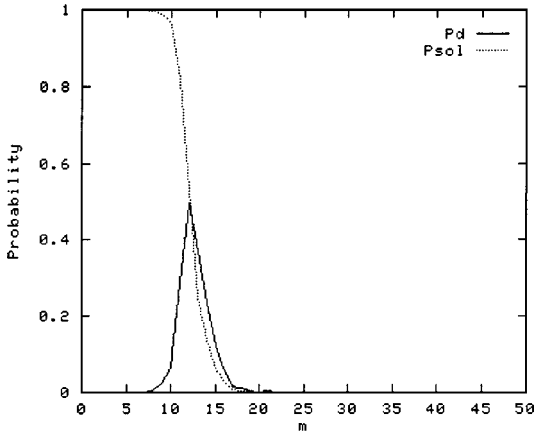


Figure 14. Proportion of hypotheses discriminating among two concept instances. For each m value, 1,000 formulas have been generated, corresponding to 2,000 matching problems. The largest fraction of discriminant hypotheses corresponds to 50% chance of a solution existing.

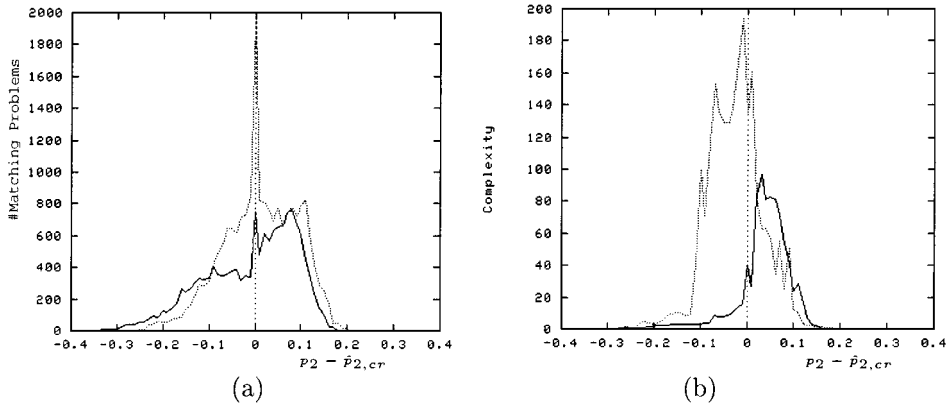


Figure 15. Evolution of the population of inductive hypotheses manipulated by G-Net. (a) Distribution of the $(p_2 - \hat{p}_{2,\mathrm{cr}})$ values corresponding to hypotheses belonging to an initial population (continuous line), and to the population after 10,000 hypothesis generation steps (dashed line). The concentration of individuals towards the phase transition clearly emerges. (b) Distribution of the matching complexity for the same populations as in (a). A remarkable increase in the matching complexity appears.

bution of the variable $(p_2 - \hat{p}_{2,\mathrm{cr}})$ for matching problems obtained by pairing each example with all the hypotheses belonging to an initial (almost random) population,[7] and the same distribution for the population reached after 10,000 hypothesis generation steps. Clearly, as time goes on, the hypotheses evolved by G-Net tend to accumulate around the phase transition point, where $p_2 = \hat{p}_{2,\mathrm{cr}}$. Figure 15(b) reports the corresponding measured matching complexity, averaged over all problems corresponding to the same $(p_2 - \hat{p}_{2,\mathrm{cr}})$ value.

The computational problems due to the matching complexity were known since long; in fact, relational learners usually set strong biases on the hypothesis description language

to control this complexity (Kietz & Wrobel, 1992; Kietz & Morik, 1994; Ade, Raedt, & Bruynooghe, 1995; desJardins & Gordon, 1995). For instance, two well investigated biases in ILP (Muggleton, 1992) are *determinacy* and *depth* (Muggleton, 1992). A literal P is said *determinate* with respect to a formula φ and a universe U, if the formula $\varphi \wedge$ P has at most the same number of models in U as φ. When hypotheses are generated incrementally by adding literals one at a time, as in FOIL, determinacy may be required for each newly added literal. The *depth* of a variable x is the number of previous variables, occurring inside the ordered body of a clause, on which the binding of x depends. Determinacy and depth can be combined, to define *ij-determinacy* (Muggleton, 1992). Imposing determinacy limits both the complexity of the hypothesis verification process, and the size of the hypothesis space, because many hypotheses are excluded, depending on the structure of the examples in the learning set.

Some formal results, related to various ILP biases, have been obtained within the PAC-learnability framework (Valiant, 1984). For instance, Džeroski, Muggleton, and Russell (1992) showed that non-recursive, constant-depth, determinate clauses are PAC-learnable. This result was extended by Cohen (1993) to linear, closed, recursive, constant-depth determinate clauses. Also, ij-clausal theories were proved to be PAC-learnable by De Raedt and Džeroski (1994). A review of decidability and complexity results related to ILP is provided by Kietz and Džeroski (1994).

The problem of taming the complexity of relational learning has also been handled in approaches different from ILP. For instance, the system G-Net (Anglano et al., 1997, 1998) exploits a *template*, which defines the syntactically most complex formula allowed in the hypothesis language. The idea of a template is also employed in the system MOBAL (Morik, 1991). Zucker (1996) introduces a hierarchy of nested languages with increasing complexity, and tries to learn starting from the simplest one. A different approach, based on stochastic sampling with polynomial complexity, is proposed by Sebag and Rouveirol (1997, 2000), who trade precision for complexity reduction. PAC-learnability, as well as classical complexity theory, is based on a worst-case analysis of a task. As we have already shown in the previous sections, not every single problem instance shows the exponential complexity characterizing the class. Requiring polynomial complexity on a whole class of problems, as in the PAC-learnability framework, has the consequence that many hypothesis description languages must be excluded from consideration, potentially hindering interesting hypotheses from being discovered. For instance, if all the literals in a clause must be determinate, the branching factor of any node in the search tree becomes upper-bounded by 1. This constraint may be too strong, and the hypothesis space may become quickly empty with the increase of the concept instance complexity.

For the above reasons, we suggest a different approach. Instead of uniformly limiting the expressiveness of the hypothesis description language, we only exclude from consideration those individual hypotheses that show an excessive matching complexity, according to an early on-line estimation. The approach, introduced in the next section, share the basic ideas with the system STILL (Sebag & Rouveirol, 1997, 2000), which already proved to be successful in learning relational concepts. Actually, STILL's sampling-based heuristic is shown to be a special case of the method introduced in this paper.

6. On-line complexity estimation

From the analysis we presented in the previous section, it appears that hypotheses built up by a learner in FOL will lie in the mushy region and may be either simple to verify or very complex. In fact, matching problems inside the phase transition show a high variability with respect to the search complexity, and apparently similar ground instances and formulas may happen to be easy to match or intractable. This consideration suggested us to complement the static analysis based on an estimate of $\hat{p}_{2,\mathrm{cr}}$ with a new procedure for recognizing on-line tractable matching problems. The basic idea is to use the stochastic search algorithm described in Section 3.1 to dynamically monitor some useful parameters. The reason for choosing this algorithm is that it is easy to analyze in a Monte Carlo framework and provides the necessary random exploration of the solution space. Nevertheless, even if in the following we will show that it performs reasonably well, it is not particularly efficient and, probably, it is possible to do better by exploiting the work made by other authors in the area of searching algorithms (Bayardo & Schrag, 1997; Gomes, Selman, & Kautz, 1998).

6.1. Search with replacement

As discussed in Section 3.1, algorithm MC can repeatedly run both with and without replacement. Even though the actual search is performed without replacement, we start our analysis with a search with replacement, which can be precisely dealt with by the theory of Monte Carlo algorithms. The entities that we are interested in are two probabilities, namely:

- An estimate $\hat{P}_{\mathrm{err}}$ of the probability of error P_{err} on solvable problems, i.e., the probability that the algorithm MC(τ, n) returns NO, in a single run, when there are indeed solutions to the problem.
- An exact upper bound, P_{Max}, of the probability of success $P_{\mathrm{succ}} = 1 - P_{\mathrm{err}}$.

When a matching problem has no solution, MC is always correct, because it will always stop with NO. Moreover, algorithm MC is *consistent*, because it never returns two different correct solutions to a same problem instance, and also *YES-biased*, because the answer y = YES is always correct, whereas the answer y = NO may be wrong. Finally, MC is $(1 - P_{\mathrm{err}})$-correct, as $(1 - P_{\mathrm{err}})$ is the probability of obtaining a correct answer on any solvable instance (Brassard & Bratley, 1988). Consistent Monte Carlo algorithms have the property that their probability of giving a correct answer increases by accepting as an answer the most frequent output in repeated runs, provided that $P_{\mathrm{err}} < 1/2$. However, for biased consistent algorithms, the same effect can be obtained even though $P_{\mathrm{err}} \geq 1/2$, provided that $P_{\mathrm{err}} < 1$. In particular, if MC is a consistent, (1-P_{err})-correct and YES-biased Monte Carlo algorithm, the algorithm obtained by letting MC run independently r times on the same instance is still a consistent and YES-biased Monte Carlo algorithm, and, in addition, it is (1-P_{err}^r)-correct.

Let MC run a generic number r of times on the same instance, and that a sequence of r NO is returned. The greater r, the more willing we would be to conclude that the problem under analysis has no solution. Actually, by exploiting the above mentioned property of Monte Carlo algorithms, we obtain an estimation of the probability that the answer is actually wrong, i.e., that $N_{\text{sol}} \geq 1$. In fact, under the hypothesis that $N_{\text{sol}} \geq 1$, a sequence of r NO has a probability to occur as low as P_{err}^r. We may conclude that, the longer the sequence of NO, the higher is the likelihood that the probability of success P_{succ} is low. More precisely, when $P_{\text{err}}^r \leq \epsilon$, we have:

$$\hat{P}_{\text{succ}} = 1 - \sqrt[r]{\epsilon} \tag{10}$$

By choosing $\epsilon = 0.0001$, expression (10) gives $\hat{P}_{\text{succ}} = 0.01$ for $R = 922$.

6.2. *Sampling without replacement*

By sampling with replacement, expression (10) provides the estimate $\hat{P}_{\text{succ}}$, but the sampling process may require a number of trials approaching infinity to find a solution, when the true value P_{succ} is greater than 0 but very small. Hence, we prefer to use the MC version without replacement. In this case, the search process always terminates in a finite number of steps. Actually, sampling with and without replacement show significant differences in the estimate only for values of P_{succ} close to zero (experimentally, $P_{\text{succ}} < 0.2$). In the case of no replacement, estimate (10) is a pessimistic one, because, at each subsequent trial, the probability of finding a solution, given that there is one, increases; then, the actual probability of success should have been lower than the one provided by (10).

Let us now consider the set of leaves of τ, i.e., τ's frontier. Let Φ_k be the set of leaves at level k in the tree, and let $m_{k+1}(\nu_k)$ be ν_k's number of sons. To each leaf ν_k (of level k), a *polychotomic fraction* $q(\nu_k)$ is associated (Watanabe, 1969):

$$q(\nu_k) = \prod_{j=1}^{k} \frac{1}{m_j(\nu_{j-1})} \tag{11}$$

The value $q(\nu_k)$ is the product of the number of sons of each node encountered along the path from the root to the node itself. By referring to the algorithm in Section 3.1, m_j is the cardinality of the set C_j. It is immediate to see (Watanabe, 1969) that:

$$\sum_{k=1}^{n} \sum_{\nu_k \in \Phi_k} q(\nu_k) = 1 \tag{12}$$

Expression (12) states that the sum of the polychotomic fractions over the frontier of τ is normalized to 1. If the frontier changes, the q's become automatically renormalized to 1. The value $q(\nu_k)$ represents the actual probability that MC outputs leaf ν_k as a result in a

single run. When the set Σ of solutions is not empty, we have:

$$P_{\text{succ}} = \sum_{\nu_k \in \Sigma} q(\nu_k) \tag{13}$$

When sampling is performed with replacement, the values of the q's do not change from one trial to another, and so P_{succ} does not change as well, whereas the q's do change in the case of sampling without replacement. If we delete from the tree the unsuccessful leaves already explored, the stochastic searcher may have to explore the whole tree before deciding that there are no solutions. In sampling without replacement, the probability P_{succ} of finding a solution in any single trial may vary from one run to another: specifically, it is monotonically non decreasing. Notice that P_{succ} does not necessarily increases in every run. In fact, let ν_k be a leaf of level k ($1 \leq k \leq n$) in which MC stops. Let $q(\nu_k)$ be its polycothomic fraction. If ν_k is removed from the search tree, its polychotomic fraction (and, in this case, its probability of being reached again) becomes zero. Then, the m_k value associated with its father ν_{k-1} decreases by 1, and the polychotomic fractions of the nodes that have ν_{k-1} as an ancestor increases. If the nodes corresponding to solutions are not descendants of ν_{k-1}, their q's values do not change. As a consequence, the probability of finding a solution may not increase at each run; however, it is bound to increase on the average over several runs.

The above considerations can be used to upper-bound P_{succ}. In fact, before starting any exploration, we do not know anything about the search tree. For instance, we do not know whether the leaves are solutions or not; then we may suppose to be in the optimistic case in which all the leaves are at level n, and so, all are solutions. Then, in the complete ignorance, we may assume $\Phi_k = 0$ for each $k \neq n$ and $P_{\text{succ}} = \sum_{\nu_n \in \Phi_n} q(\nu_n) = 1$, i.e., all leaves are solutions, and MC will certainly find one at the first run. When we perform a first trial, which ends in a non-solution leaf $\nu^{(1)}$, we know with certainty that the probability P_{succ} was actually no greater than $[1 - q(\nu^{(1)})]$. By performing other unsuccessful trials, each time the upper bound of P_{succ} decreases by the polychotomic fraction of the last found leaf $\nu^{(r)}$. After R trials:

$$P_{\text{succ}} \leq 1 - \sum_{r=1}^{R} q\left(\nu^{(r)}\right) \stackrel{\text{def}}{=} P_{\text{Max}} \tag{14}$$

Theoretically computing a reasonable approximation of P_{Max} is hard. Then, we evaluate P_{Max} on-line, deciding, after R unsuccessful trials, whether we are willing to accept the NO answer as the correct one, with a preset probability of being mistaken, or we want to continue the search.

6.3. *Experimental evaluation*

In the following, let us define $P = 1 - \sqrt[R]{\epsilon}$. Then:

$$P \approx P_{\text{succ}} \leq P_{\text{Max}} \tag{15}$$

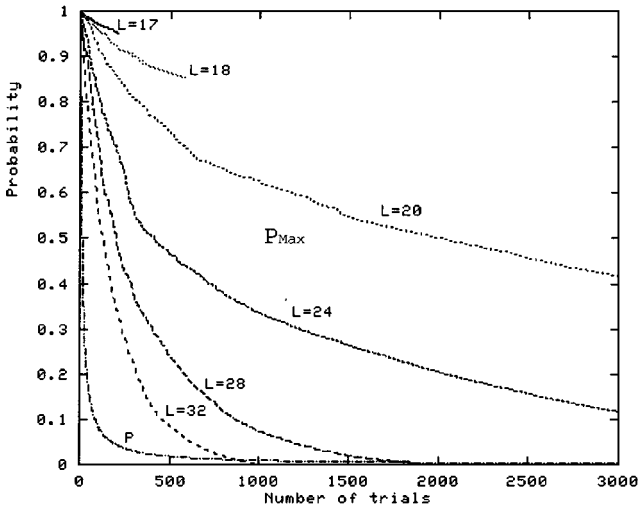


Figure 16. Temporal evolution of P and P_{Max} in a formula φ with 10 variables, and 19 binary predicates, for different values of L. By choosing $\epsilon = 0.0001$, $\hat{P}_{\text{succ}}$ reaches 0.01 for $R = 922$.

We may notice that P only depends upon ϵ and R, whereas P_{Max} depends upon the structure of the particular search tree. By increasing R, P_{Max} should converge to P. In order to show how P and P_{Max} can be used, we performed an experimental analysis on a subset of the formulas used in Section 3. The results are exemplified in figure 16, which describes typical time evolutions of P_{Max} and P for a formula φ with 10 variables and 19 literals, selected as representative of the set. From figures 2 and 3, we see that a formula with $n = 10$ and $m = 19$ undergoes a phase transition for a value of L between 20 and 25, when $N = 100$. For $L < 20$ the matching problems are almost always solvable and easy, and for $L > 25$ the matching problems are almost always unsolvable and easy.

In figure 16 we observe that the behavior of P_{Max} is very different in the three regions, with respect to its derivative: when the problems are solvable, the rate of descent of P_{Max} is low, but the curve stops early because a solution is easily found. When problems are unsolvable, the rate of descent of P_{Max} is high, and again the search stops quickly, because it is easy to prove unsolvability (the search tree is small). Finally, inside the phase transition region, P_{Max} decreases slowly and we may need excessive computational resources to arrive at a conclusion. It may be advisable, in this case, to give up searching, and to accept a NO answer as the correct one. The graphs of figure 16 confirm the results reported by Walsh (1998), who showed that difficult problems at the phase transition remain difficult as search proceeds.

As the rate of decay is similar on the left of and inside the mushy region at the beginning of the search, it may not be possible to predict very early which of the two cases actually is the current one, on the basis of P_{Max} only. We combine then the information from both P_{Max} and P.

When P_{Max} decreases slowly and P predicts a very low probability of success, we can assume that the matching problem we are handling is hard, probably close to the phase

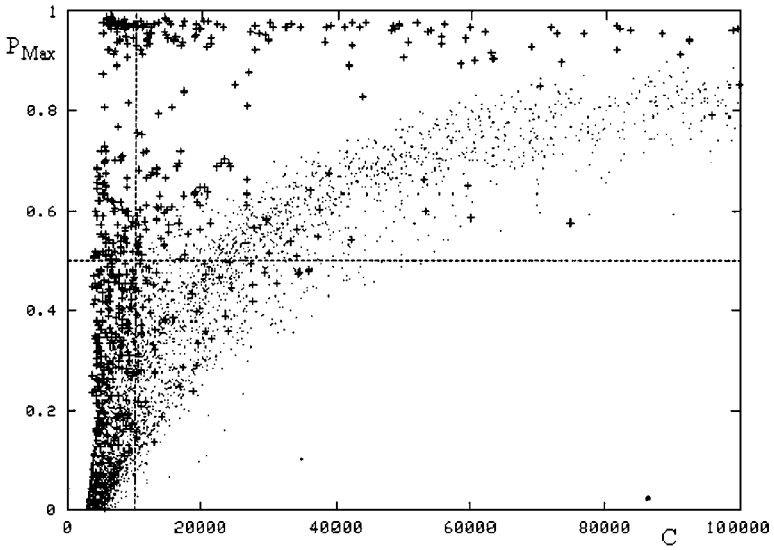


Figure 17. P_{Max} values measured at $P = 0.01$ (R = 922 trials, with $\epsilon = 0.0001$) versus the total number C of visited nodes. Symbols "+" and "·" denote solvable and unsolvable matching problems, respectively.

transition. To test this hypothesis, we have performed a set of experiments, whose results are reported in figure 17. We have generated 25,000 matching problems, with $n = 10$, $m = 19$, and L varying between 11 and 50, in order to cross the mushy region in the maximum complexity zone. For each problem, MC ran repeatedly, without replacement, until either a solution was found or the whole tree was visited without finding a solution. Let C be the total number of nodes visited by MC during the search on a given problem. C is the complexity of the search, and is reported on the horizontal axis of figure 17.

Let R denote the value of r at which we suspend the search. For all the matching problems still undecided at R, let us measure the corresponding $P_{\mathrm{Max}}(R)$, and report this value on the vertical axis of figure 17. Then, each point $(C_i, P_{\mathrm{Max},i})$ corresponds to a particular problem π_i, which has the following characteristics:

(a) π_i is still undecided after R trials with MC.
(b) The estimated probability of error, if we accept a NO answer, is less than ϵ, and the probability of success in any single trial should not have exceeded 0.01.
(c) The probability P_{succ} is exactly upper bounded by $P_{\mathrm{Max},i}$.
(d) π_i required C_i steps to arrive to a precise determination of its solvability.

Notice that problems that were decided at some $r < R$ do not appear in the figure, and, then, only the problems inside the complexity peak have been considered. If another snapshot would be taken at a greater R, a downward shift of the points would be observed in figure 17; moreover, some points could disappear, because they will get a precise answer.

Therefore, given two thresholds, θ_1 on P and θ_2 on P_{Max}, respectively, a simple criterion for deciding whether to stop the matching process at $r = R$ can be captured by the following rule:

"*If* $P \leq \theta_1$ *and* $P_{\text{Max}} \geq \theta_2$, *Then stop the matching process*" (16)

The effect of rule (16) can be visualized in figure 17 by drawing a horizontal line corresponding to a threshold θ_2. For all the problems corresponding to points above the line the matching process will be interrupted when $P = 0.01$. As we can see, the maximum measured complexity increases very quickly when $P_{\text{Max}} \geq 0.5$.

The results of a more accurate analysis of the effects of θ_2 on the performances of the stochastic matching are reported in Table 3, where the upper part refers to the problems in the range $15 \leq L \leq 40$ (i.e., the whole peak), whereas the lower part refers to problems located very close to the critical point ($17 \leq L \leq 24$). The second column contains the threshold value for θ_2, corresponding to the estimated probability of success $\hat{P}_{\text{succ}} = P = 0.01$ with a reliability $(1 - \epsilon) = 0.9999$. We recall that all the matching processes for which $P_{\text{Max}} \geq \theta_2$ at $r = R$ will be stopped. What to do with them is up to the user: they may be declared "undecided", increasing the number the cases in which the resulting classifier does not give an answer, or they may be declared unsatisfiable, possibly increasing the number of errors on satisfiable examples. Setting $\theta_2 = 0$ means that every matching process stops as soon as P reaches the value 0.01. Setting $\theta_2 = 1$ means that no matching process is stopped. The third column contains the average complexity evaluated on all the problems, including

Table 3. Maximum and average matching complexity, in the region of the phase transition induced by the number L of constants in the universe. The results are reported separately for solvable (S) and unsolvable (U) problem instances. The stop of the matching process has been decided when $P = 0.01$ with reliability $1 - \epsilon = 0.9999$.

1	2	3	4	5	6	7	8	9	10	11
Range of L	θ_2	C_{Avg}	C_{Wst}	% Decided problems	$C^{(S)}_{\text{Max}}$	$C^{(S)}_{\text{Avg}}$	$C^{(U)}_{\text{Max}}$	$C^{(U)}_{\text{Avg}}$	S	U
[15, 40]	0.0	2335.9	1607.8	0.651	5439	704.2	4579	1353.6	0.841	0.577
	0.3	2687.2	968.8	0.789	19235	926.0	27008	2737.2	0.872	0.757
	0.4	3007.5	809.9	0.824	24135	1080.7	36399	3355.2	0.889	0.798
	0.5	3471.3	661.3	0.856	35863	1320.3	52417	4115.1	0.910	0.835
	0.6	4152.4	516.3	0.888	74924	1589.7	90145	5141.8	0.932	0.870
	1.0	15975.5	0.0	1.000	340969	3682.7	1170012	20774.3	1.000	1.000
[17, 25]	0.0	2573.2	1933.8	0.580	5439	998.9	4579	1903.2	0.782	0.192
	0.3	3102.6	1283.8	0.721	19235	1299.8	27008	6200.9	0.823	0.525
	0.4	3524.7	1082.2	0.765	24135	1514.5	36399	7692.8	0.847	0.607
	0.5	4088.5	877.0	0.809	35863	1845.1	52417	9224.0	0.877	0.680
	0.6	4606.3	722.6	0.843	74924	2212.1	90145	10391.6	0.906	0.721
	1.0	29354.0	0.0	1.000	340969	5040.5	1170012	76132.5	1.000	1.000

the ones that terminate before reaching the step $R = 922$, and the ones which have been interrupted. The fourth column contains the computational cost, averaged on all matching problems, which has been wasted for the problems interrupted after reaching $P = \theta_1$. The fifth column contains the percentage of problems which have not been interrupted, i.e., which have been proved solvable or unsolvable. The sixth and eighth columns contain the maximum experimental complexity, measured separately for solvable and unsolvable problems. This maximum complexity corresponds to the abscissa of the rightmost point (for solvable and unsolvable instances, separately) occurring under the horizontal line $P_{\text{Max}} = \theta_2$. The seventh and ninth columns contain the average global complexity required for $1 \leq r \leq R$, for solvable and unsolvable problems, respectively. Finally, the tenth (eleventh) column contains the fraction of solvable (unsolvable) problems among the ones that would run to completion if the threshold θ_2 is chosen when $P = 0.01$. This fraction can be evaluated by the number of solvable (unsolvable) problems whose corresponding points lay under the line $P_{\text{Max}} = \theta_2$ in figure 17, augmented by the number of solvable (unsolvable) problems that stopped before R steps, divided by the total number of solvable (unsolvable) problems.

We can see from Table 3 that, by choosing $\theta_2 = 0.5$, the maximum complexity for running to completion about 86% of the problems (Table 3, columns 5) is less than $1/10$ of the maximum complexity over all solvable problems (Table 3, columns 6), and less than $1/20$ over all unsolvable problems (Table 3, column 8).[8] This means that all the extremely hard instances are cut away. Remarkable reductions are also obtained for the average complexity (Table 3, columns 7 and 9). An optimal combination of the threshold values on P and P_{Max} could be experimentally found.

A simpler rule to limit the complexity would be to stop the matching process as soon as the probability P reaches θ_1. Threshold θ_1 can be lowered in order to allow a sufficient exploration of the solution space. Table 4 reports the complexity values and the fraction of perfectly answered problems for different stopping values of θ_1, ranging from 0.01 to 0.001. In Table 4 the columns have the same meaning as in Table 3. By comparing Table 3 and Table 4, it appears that, for comparable average complexities, the fraction of problems precisely answered using rule (16) is significantly higher. For instance, by considering a threshold $\theta_2 = 0.5$, we obtain an average complexity of 3471 steps (Table 3, column 2), which is a little less than the average complexity found by setting $\theta_1 = 0.005$ (Table 4, column 2). Nevertheless, the fraction of problems run to completion is about 86% in the first case (Table 4, column 5), while it is only 77% in the second one (Table 4, column 5). As an alternative, a percentage of 87% of completed problems can be obtained by setting $\theta_1 = 0.002$ (Table 4, column 5), but in this case the average complexity would be more than 5783 steps (Table 4, column 2).

Considering the lower parts of Tables 3 and 4, we observe that getting closer to the critical point, the difference between the two stopping rules increases. Finally, considering the last two columns in Tables 3 and 4, we see that the fraction of problems run to completion has a different composition in the two cases. More specifically, using rule (16) we have a greater percentage of problems proved unsolvable (Table 3, column 11, and Table 4, column 11) and a smaller percentage of problem proved solvable (Table 3, column 10, and Table 4, column 10). The reason can be understood by observing that the fraction of problems interrupted by setting a threshold on θ_1 is represented by the points in figure 17

Table 4. Maximum and average matching complexity, in the region of the phase transition induced by the number L of constants in the universe. The results are reported separately for solvable (S) and unsolvable (U) problem instances. The matching process is halted when $P \leq \theta_1$, with $1 - \epsilon = 0.9999$.

1	2	3	4	5	6	7	8	9	10	11
Range of L	θ_1	C_{Avg}	C_{Wst}	% Decided problems	$C^{(S)}_{\text{Max}}$	$C^{(S)}_{\text{Avg}}$	$C^{(U)}_{\text{Max}}$	$C^{(U)}_{\text{Avg}}$	S	U
[15, 40]	0.010	2335.9	1607.8	0.651	5439	704.2	4579	1353.6	0.841	0.577
	0.005	3503.6	2111.3	0.771	10066	1178.1	10495	2122.2	0.920	0.712
	0.003	4647.7	2570.7	0.832	17021	1578.9	19621	2932.2	0.956	0.783
	0.002	5783.0	3055.6	0.867	24135	1835.4	26419	3746.1	0.970	0.826
	0.0015	6710.0	3321.9	0.891	31106	2017.0	32369	4592.4	0.977	0.858
	0.001	8155.3	3691.0	0.919	51783	2371.1	49534	5925.8	0.986	0.893
[17, 24]	0.010	2573.2	1933.8	0.580	5439	998.9	4579	1903.1	0.782	0.192
	0.005	4112.8	2537.8	0.724	10066	1633.5	10495	4442.3	0.889	0.407
	0.003	5403.3	2698.0	0.824	17021	2174.5	19621	6624.6	0.940	0.600
	0.002	6611.0	3319.8	0.855	24135	2522.6	26419	7575.3	0.959	0.655
	0.0015	7666.7	3953.8	0.871	31106	2769.8	31800	8339.0	0.968	0.682
	0.001	9503.7	5006.9	0.890	51783	3238.1	49369	9812.8	0.981	0.717

lying to the right of a vertical line corresponding to the maximum complexity found before reaching θ_1 (Table 4, columns 6 and 8). The vertical dotted line in figure 17 corresponds to $\theta_1 = 0.005$.

Then, rule (16) offers a good criterion for avoiding to be trapped in an excessively costly matching process inside the mushy region. An obvious way of using rule (16) in a learning algorithm consists in rejecting all the inductive hypotheses that are not provable either true or false within assigned number of steps. This heuristics is easy to be included in any learning algorithm.

Furthermore, still weaker biases are possible. For instance, we notice from figure 17 and Tables 3 and 4 that negative examples usually exhibit higher complexity than positive ones. This means that, if we consider unsatisfiable a hypothesis stopped by rule (16), we may make a mistake. However, if the number of trials is large enough, the proportion of these mistakes may be of the same order of magnitude as the error due to noise in typical real-world applications.

This last observation is exploited by the system STILL (Sebag & Rouveirol, 1997, 2000). STILL makes use of a stop criterion based on the only estimate of the error probability P. When P decreases below a given threshold the matching stops; this criterion is equivalent to set $\theta_2 = 0$ in rule (16). With respect to STILL's criterion, rule (16) with $\theta_2 > 0$ allows a smaller error rate to be achieved for the same average complexity, or, alternatively, the same precision to be reached by paying a smaller computational cost.

Other proposals of using stochastic sampling for estimating parameters relevant to search have been presented by Frost, Rish, and Vila (1997), Huberman, Lukose, and Hogg (1997), Bailleux and Chabrier (1996), and Bailleux (1998).

7. Conclusions

The recent literature in Machine Learning and Data Mining shows a growing interest towards applications of relational learning to knowledge extraction in domains characterized by highly structured data, such as Chemistry or Molecular Biology. If, on the one hand, description languages based on First Order Logics offer an important improvement to deal with structured data, on the other hand, the high complexity hidden in the hypothesis verification step challenges the chances of success of relational learning on large scale applications. In fact, relational learners have been proved successful, so far, only on simple tasks, in which hypotheses had to obey to strong syntactic and/or semantic biases.

In this paper, we tried to trace back at least one of the possible sources of the complexity in relational learning, namely hypothesis verification. The emerging findings suggest that there may be severe scalability problems in inductive approaches to relational learning, as soon as applications requiring descriptions with many variables are faced. New heuristics should be devised, capable of "distracting" the learner from the attraction of the phase transition. A possible way out may be to use domain specific knowledge.

The method proposed in Section 6 does not offer a way of keeping the learner away from the phase transition region. However, it does offer the benefit of reducing the amount of likely useless search, without constraining too much the hypothesis space. The same is true for other approaches to improve efficiency in FOL learning, such as caching previous expensive computations or memorizing partial evaluations, as proposed, for instance, by Pompe (1996), or implemented in the P-Progol version of Progol (Muggleton, 1995).

The empirical results reported both in this paper and by other authors (Sebag & Rouveirol, 1997) suggest that stochastic sampling can be a viable approach. To this aim the literature related to searching algorithms provides suggestions worth to investigate (Gomes, Selman, & Kautz, 1998; Brassard & Bratley, 1988).

Acknowledgments

We would like to thank Michele Sebag for the careful reading of the paper and for several useful suggestions, and Marco Botta for many fruitful discussions and for providing the stochastic matcher used for the experiments reported here.

Notes

1. Different ordering of the variables, both static and dynamic, have been tried, without noticing changes in the emergence of the phase transition.
2. During the generation of the graph in figure 3(a) (160.000 matching problems), S_2 never exceeded the memory size of 2 Mbytes, whereas S_1 grew up to 12 Mbytes. The elapsed time was about 215 min for S_1 and 280 min for S_2. When processing formulas with 14 variables, several times S_1 was unable to finish, whereas S_2 exhibited a typical size of 2 Mbytes in the mushy region, and reached a maximum of 56 Mbytes. The elapsed time was 1650 min for S_2 whereas S_1 ran for several days due to intensive use of the virtual memory.
3. The dataset used here is the "regression friendly" one: it includes those examples that can be modeled with a good approximation by linear regression.
4. A discussion on the relations between binary and non binary CSPs is provided by Bacchus and van Beek (1998).

5. In these experiments the whole dataset has been used, because we are not interested, here, in evaluating the predictive power of the learned knowledge, but only the impact of the matching complexity on learning.
6. In the real-world application, the system ENIGMA was used (Giordana et al., 1993), but now we re-analyzed the dataset with the new system G-Net. In fact, the knowledge base used in-field was obtained with an integration of SBL and EBL, and was a structured knowledge base with chains of disjunctive rules, instead of flat ones. In the cited paper, a complete description of the application can be found.
7. G-Net uses a special *seeding* operator to generate the initial population of hypotheses. Details of the procedure can be found in Anglano et al. (1998).
8. Considering the implementation S_1 of the stochastic algorithm (see figure 3(c)), the CPU time (in centi-seconds) can be found by multiplying by 0.0015 the tree size. Considering implementation S_2, we see that it may be a reasonable option, because the proposed strategy stops the search when the contribution of the quadratic term becomes significant.

References

Ade, H., Raedt, L. D., & Bruynooghe, M. (1995). Declarative bias for specific-to-general ILP systems. *Machine Learning, 20*, 119–154.

Anglano, C., Giordana, A., Lo Bello, G., & Saitta, L. (1997). A network genetic algorithm for concept learning. In *Proceedings of the 7th International Conference on Genetic Algorithms* (pp. 434–441). MI: East Lansing.

Anglano, C., Giordana, A., Lo Bello, G., & Saitta, L. (1998). An experimental evaluation of coevolutive concept learning. In *Proceedings of the 15th International Conference on Machine Learning* (pp. 19–23). Madison, WI.

Bacchus, F. & van Beek, P. (1998). On the conversion between non-binary and binary constraint satisfaction problems. In *Proceedings of the 15th National Conference on Artificial Intelligence* (pp. 311–318). Madison, WI.

Bailleux, O. (1998). Local search for statistical counting. In *Proceedings of the 15th National Conference on Artificial Intelligence* (pp. 386–391). Madison, WI: Morgan Kaufman.

Bailleux, O. & Chabrier, J.-J. (1996). Approximate resolution of hard numbering problems. In *Proceedings of the 13th National Conference on Artificial Intelligence* (pp. 169–174). Portland, Oregon: AAAI-Press.

Bayardo, R. & Schrag, R. (1997). Using CSP look-back techniques to solve real-world SAT instances. In *Proceedings of the 14th National Conference on Artificial Intelligence* (pp. 203–208). Providence, RI.

Bergadano, F. Giordana, A., & Saitta, L. (1988). Learning concepts in noisy environment. *IEEE Transaction on Pattern Analysis and Machine Intelligence, PAMI-10*, 555–578.

Botta, M. Giordana, A., & Saitta, L. (1999). Relational learning: Hard problems and phase transitions. In *Proceedings of the 16th International Joint Conference on Artificial Intelligence* (pp. 1198–1203). Stockholm, Sweden.

Brassard, G. & Bratley, P. (1988). *Algorithmics: Theory and Practice*. NJ: Prentice Hall, Englewood Cliffs.

Cheeseman, P., Kanefsky, B., & Taylor, W. (1991). Where the really hard problems are. In *Proceedings 12th International Joint Conference on Artificial Intelligence* (pp. 331–337). Sidney, Australia.

Cohen, W. (1993). A PAC-learning algorithm for a restricted class of recursive logic programs. In *Proceedings of the 10th National Conference on Artificial Intelligence* (pp. 86–92). Washington, DC.

Crawford, J. & Auton, L. (1996). Experimental results on the crossover point in random 3-SAT. *Artificial Intelligence, 81*, 31–58.

De Raedt, L. & Džeroski, S. (1994). First-order jk-clausal theories are PAC-learnable. *Artificial Intelligence, 70*, 375–392.

desJardins, M. & Gordon, D. (Eds.). (1995). *Machine Learning: Special Issue on Bias Evaluation and Selection* (*Vol. 20*). Kluwer Academic.

Džeroski, S., Muggleton, S., & Russell, S. (1992). PAC-learnability of determinate logic programs. In *Proceedings of COLT-92* (pp. 128–134). Pittsburgh, PA.

Freeman, J. (1996). Hard random 3-SAT problems and the Davis-Putnam procedure. *Artificial Intelligence, 81*, 183–198.

Frost, D., Rish, I., & Vila, L. (1997). Summarizing CSP hardness with continuous probability distributions. In *Proceedings of the 14th National Conference on Artificial Intelligence* (pp. 327–333). Providence, RI.

Gent, I. & Walsh, T. (1995). Phase transitions from real computational problems. In *Proceedings of the 8th International Symposium on Artificial Intelligence* (pp. 356–364).

Gent, I. & Walsh, T. (1996). The TSP phase transition. *Artificial Intelligence, 88*, 349–358.

Giordana, A., Neri, F., Saitta, L., & Botta, M. (1997). Integrating multiple learning strategies in first order logics. 1 *Machine Learning, 27*, 209–240.

Giordana, A., Saitta, L., Bergadano, F., Brancadori, F., & De Marchi, D. (1993). ENIGMA: A system that learns diagnostic knowledge. *IEEE Transactions on Knowledge and Data Engineering, KDE-5*, 15–28.

Giordana, A., Saitta, L., Sebag, M., & Botta, M. (2000). Analyzing relational learning in the phase transition framework. In *Proceedings of the 17th International Conference on Machine Learning* (pp. 311–318), Stanford, CA.

Gomes, C. & Selman, B. (1997). Problem structure in the presence of perturbations. In *Proceedings of the 14th National Conference on Artificial Intelligence* (pp. 431–437). Providence, RI.

Gomes, C., Selman, B., & Kautz, H. (1998). Boosting combinatorial search through randomizattion. In *Proceedings of the 15th National Conference on Artificial Intelligence* (pp. 431–437). Madison, WI.

Hogg, T. (1996). Refining the phase transition in combinatorial search. *Artificial Intelligence, 81*, 127–154.

Hogg, T., Huberman, B., & Williams, C. (Eds.). (1996a). *Artificial Intelligence: Special Issue on Frontiers in Problem Solving: Phase Transitions and Complexity* (*Vol. 81*)(1–2). Elsevier.

Hogg, T., Huberman, B., & Williams, C. (1996b). Phase transitions and the search problem. *Artificial Intelligence, 81*, 1–15.

Huberman, B., Lukose, R., & Hogg, T. (1997). An economics approach to hard computational problems. *Science, 275*, 51–54.

Hyafil, L. & Rivest, R. (1976). Constructing optimal binary decision trees is NP-complete. *Information Processing Letters, 5*, 15–17.

Kietz, J. & Džeroski, S. (1994). Inductive logic programming and learnability. *SIGART Bulletin, 5*, 22–32.

Kietz, J. & Morik, K. (1994). A polynomial approach to the constructive induction of structural knowledge. *Machine Learning, 14*, 193–218.

Kietz, J., & Wrobel, S. (1992). Controlling the complexity of learning through syntactic and task-oriented models. In Muggleton, S. (Ed.), *Inductive Logic Programming* (pp. 107–126). Academic Press, London, UK.

Michalski, R. (1980). Pattern recognition as a rule-guided inductive inference. *IEEE Transactions on Pattern Analysis and Machine Intelligence, PAMI-2*, 349–361.

Mitchell, T. (1982). Generalization as search. *Artificial Intelligence, 18*, 203–226.

Morik, K. (1991). Balanced cooperative modeling. In *Proceedings of the 1st Multistrategy Learning Workshop* (pp. 65–80). Harpers Ferry, WV.

Muggleton, S. (Ed.). (1992). *Inductive Logic Programming*. London, UK: Academic Press.

Muggleton, S. (1995). Inverse entailment and Progol. *New Generation Computing, 13*, 245–286.

Pompe, U. (1996). Efficient proof encoding. *Lecture Notes in Artificial Intelligence, 1314*, 299–314.

Prosser, P. (1996). An empirical study of phase transitions in binary constraint satisfaction problems. *Artificial Intelligence, 81*, 81–110.

Quinlan, R. (1990). Learning logical definitions from relations. *Machine Learning, 5*, 239–266.

Sebag, M. & Rouveirol, C. (1997). Tractable induction and classification in first order logic via Stochastic Matching. In *Proceedings of the 15th International Joint Conference on Artificial Intelligence* (pp. 888–893). Nagoya, Japan.

Sebag, M. & Rouveirol, C. (2000). Stochastic relational inference: Sampling-based heuristics for any-time inductive and deductive reasoning. *Machine Learning, 38*, 41–62.

Selman, B. & Kirkpatrick, S. (1996). Critical behavior in the computational cost of satisfiability testing. *Artificial Intelligence, 81*, 273–296.

Smith, B. & Dyer, M. (1996). Locating the phase transition in binary constraint satisfaction problems. *Artificial Intelligence, 81*, 155–181.

Srinivasan, A., Muggleton, S., & King, R. (1995). Comparing the use of background knowledge by two ILP systems. In *Proceedings of the 5th International Workshop on ILP* (pp. 199–229). Leuven, Belgium.

Valiant, L. (1984). A theory of the learnable. *Communications of the ACM, 27*, 1134–1142.

Walsh, T. (1998). The constrainedness knife-edge. In *Proceedings of the 15th National Conference on Artificial Intelligence* (pp. 406–411). Madison, WI.

Watanabe, S. (1969). *Knowing and Guessing: A Quantitative Study of Inference and Information*. New York, NY: John Wiley & Sons.

Williams, C. & Hogg, T. (1994). Exploiting the deep structure of constraint problems. *Artificial Intelligence, 70*, 73–117.

Zhang, W. & Korf, R. (1996). A study of complexity transition on the asymmetric travelling salesman problem. *Artificial Intelligence, 81*, 223–239.

Zucker, J.-D. (1996). Representation changes for efficient learning in structural domains. In *Proceedings of the 13th International Conference on Machine Learning* (pp. 543–551). Bari, Italy.

Received November 9, 1998
Revised November 29, 1999
Final manuscript November 29, 1999