Phase Transitions in Machine Learning

Lorenza Saitta
Attilio Giordana
Antoine Cornuéjols

CAMBRIDGE

CAMBRIDGE www.cambridge.org/9780521763912

This page intentionally left blank

Phase Transitions in Machine Learning

Phase transitions typically occur in combinatorial computational problems and have important consequences, especially with the current spread of statistical relational learning and of sequence learning methodologies. In *Phase Transitions in Machine Learning* the authors begin by describing in detail this phenomenon and the extensive experimental investigation that supports its presence. They then turn their attention to the possible implications and explore appropriate methods for tackling them.

Weaving together fundamental aspects of computer science, statistical physics, and machine learning, the book provides sufficient mathematics and physics background to make the subject intelligible to researchers in the artificial intelligence and other computer science communities. Open research issues, suggesting promising directions for future research, are also discussed.

LORENZA SAITTA is Full Professor of Computer Science at the University of Piemonte Orientale, Italy.

ATTILIO GIORDANA is Full Professor of Computer Science at the University of Piemonte Orientale, Italy.

ANTOINE CORNUÉJOLS is Full Professor of Computer Science at the AgroParisTech Engineering School, Paris.

Phase Transitions in Machine Learning

LORENZA SAITTA
University of Piemonte Orientale, Italy

ATTILIO GIORDANA
University of Piemonte Orientale, Italy

ANTOINE CORNUÉJOLS
AgroParisTech Engineering School, Paris, France

CAMBRIDGE
UNIVERSITY PRESS

CAMBRIDGE UNIVERSITY PRESS
Cambridge, New York, Melbourne, Madrid, Cape Town,
Singapore, São Paulo, Delhi, Tokyo, Mexico City

Cambridge University Press
The Edinburgh Building, Cambridge CB2 8RU, UK

Published in the United States of America by Cambridge University Press, New York

www.cambridge.org
Information on this title: www.cambridge.org/9780521763912

First published 2011

Printed in the United Kingdom at the University Press, Cambridge

A catalogue record for this publication is available from the British Library

ISBN 978-0-521-76391-2 Hardback

Contents

Preface

From its inception in the 1930s, the rich and vigorous field of computer science has been concerned with the resources, both in time and in memory, needed to carry out a computation. A number of fundamental theorems were discovered that resorted to a worst-case analysis. The central question was whether a given algorithm could be guaranteed to terminate a computation in finite time whatever the inputs, and, if so, in which class of complexity it lay, given the control parameters: polynomial, exponential, and so on. Therefore, in 1991, a paper by Cheeseman, Kaneefsky, and Taylor came as a bolt from the blue. Indeed, while its title, "Where the really hard problems are", was not altogether disturbing, its content was. Broadly speaking, the authors argued that even if it was important to analyze worst cases, it was just as essential to look for the typical complexity of computations, the complexity encountered when solving typical problems. And there lies a gem: the transition from the region of problems that are hard, in terms of algorithmic complexity, to the region of problems that are easy can be quite sharp. Moreover, these regions and transitions are not related to the worst cases.

We remember that this 1991 paper, presented at the International Joint Conference on Artificial Intelligence (IJCAI), started a commotion, though how profound this would be was not at first apparent. We were among those who felt that this paper and others that promptly followed, from physicists in particular, were significant beyond the obvious. However, this event did not alter the course of machine learning, our field, for many years. In machine learning too the theoretical analysis that was at that time taking shape dealt with a type of worst-case study; this new statistical theory of learning was sweeping the field and gaining momentum as new learning algorithms, inspired in part by its lessons, were devised.

Thus, it was only in 1999 that M. Botta and two of us[1] finally published a paper that took in the new perspective opened by Cheeseman and others and

[1] Attilio Giordana and Lorenza Saitta (Botta *et al.*, 1999).

examined its impact on machine learning or, to be more specific, on the matching problem that is at the heart of learning. And here again, as with hindsight could have been suspected, a phase transition came into view. Over the following years we, and others, carried out thorough empirical investigations, which all confirmed and elaborated this finding. Even though the mainstream of machine learning was still under the spell of the statistical and worst-case-analysis perspective, it was becoming apparent that these results, which could not be accounted for by the dominant view, had a quite significant potential impact on the very feasibility of learning. Indeed, some known failures in the learning of large-scale problems could be explained thanks to this new point of view.

The fact is that, at least for some learning problems, there exists a sharp discontinuity between easy and hard matching problems. This severely hinders, at the very least, the assessment of candidate hypotheses considered during learning, therefore making the exploration of solutions all but blind. It is no wonder that the consequences can be quite serious.

While a complete understanding of the phase transition in learning still eludes us as a community of researchers, we feel that the wealth of results obtained in recent years and their known links with other fields in computer science and physics are now sufficiently mature to deserve a wide encompassing presentation, one that would describe as large a part as possible of the phase transition phenomena relevant to machine learning and would stimulate further research on this important subject. This book is the result of our conviction that the study of phase transitions in machine learning is important for the future of machine learning, and it presents us with the opportunity to establish profound connections with other natural sciences.

The book deals with the border between statistical physics, complex systems, and machine learning: it explores emergent properties in relational machine learning using techniques derived from statistical physics. More generally, the book is concerned with the emergence, in learning, of a phase transition, a phenomenon typically occurring both in many-body systems and in combinatorial problems.

This phenomenon is described in detail, and the extensive experimental investigation that supports its presence is reported. Then the results and the implications that the appearance of a phase transition may have on the scalability of relational learning and on the quality of the acquired knowledge are discussed in depth. With the current spread of statistical relational learning methodologies this topic is assuming an increasingly strong relevance.

The idea behind the book is to stimulate synergic research interests in the fields of both statistical physics and machine learning. Researchers in the former may find in machine learning a rich, appealing field, where their theories and

methods can be applied whereas researchers in the latter may find new tools for investigating and explaining learning processes in depth.

The identification of a phase transition in a computational problem may have important consequences in practice. In fact, as mentioned above, the standard notion of the computational complexity of a class of problems is a pessimistic evaluation based on a worst-case analysis. 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* complexity. Relational learning is a task particularly affected by the problem of high computational complexity. In this book, we are only concerned with supervised learning for classification, within the paradigm of *learning from examples*.

A theoretical approach, inspired by statistical physics, and a supporting set of experiments have uncovered that, in relational learning, the expected phase transition occurs inside a range of parameter values that is relevant for practical learning problems. It is thus sensible to investigate the phenomenon and to try to propose possible ways around it, since the emergence of a phase transition in relational learning can have a large negative impact on a task's feasibility.

In order to underline that the emergence of a phase transition is far from exceptional, we have widened the scope of the book to include grammar induction and an overview of related topics in neural networks and other propositional learning approaches showing the ubiquity of the phenomenon. Moving outside the machine learning area, we also describe the emergence of phase transitions in complex networks and in natural systems, including human cognition. Again, the links between the findings observed in such a variety of systems may stimulate cross-correlations and convergence.

We hope that the deep interactions that we will discuss between the theoretical issues and the experimental findings will provide a rather complete landscape of the field, including both the foundational aspects and the practical implications. Our intention is that the novelty of the topic, the analysis of foundational issues in machine learning, and our attention to practical solutions and applications will make the book appeal to a variety of readers. The detailed explanations of findings should facilitate understanding of the various viewpoints even for readers not within the field.

Even though the book mainly targets a readership familiar with artificial intelligence and machine learning, its foundational aspects will also be of interest to cognitive scientists, and even philosophers, looking for the emergence and the epistemological impact of similar phenomena in nature. The book may be of particular interest to researchers working on complex systems, as we make an explicit effort to link the phenomena investigated to the theory of these systems. Likewise, researchers in statistical physics who are interested in its computational aspects may be attracted by the book.

The approach taken is primarily quantitative and rigorous. Nevertheless, we have provided intuitive and qualitative illustrations and explanations of the issues and results. The idea is that even non-technical readers should be able to understand the main issues. For a qualitative understanding, the basic notions of artificial intelligence (especially knowledge representation and search) and computer science (especially computational complexity) are necessary. For a quantitative understanding, probability theory and advanced calculus are required.

Reading the book should allow a researcher to start work in the field without searching, reading, and linking many articles found dotted about in a variety of journals. Also, the book should be of help for those wanting to understand some of the philosophical problems underlying computation.

Above all else we would be happy to see new research themes originating from this book.

Acknowledgments

Several friends and colleagues have contributed to this book, directly or indirectly, and we are indebted to them all.

We want to thank in particular Michèle Sebag: long and pleasant discussions with her greatly contributed to our understanding and broadened our horizons; working with her was an enriching pleasure.

Also many thanks go to Erick Alphonse and Omar Osmani, who followed our first steps in the field with enthusiasm, contributing many new ideas and results.

In this book we have reported the work of many researchers, and their permission to reprint graphics from their papers has spared us a lot of additional work; hearty thanks to them as well.

Finally, we are very grateful to Enrico Scalas, a colleague physicist who carefully checked our introduction to statistical physics and took the time to provide us with detailed and insightful comments.

Notation

$\mathtt{P}$	Probability (for a finite set)
$\mathtt{p}$	Probability density
$\boldsymbol{G}$	Graph
$\mathcal{G}$	Ensemble of graphs
$\mathbb{E}[x]$	Expectation of x
$\mathbb{V}[x]$	Variance of x
$\mathcal{O}(\cdot)$	"Big O" notation: describes the limiting behavior of a function when the argument tends towards infinity
$\mathbb{R}$	The real numbers
$\mathbb{R}^n$	The space of real numbers of dimension n
$\mathbb{N}$	The natural numbers
$\mathbb{B}^n = \{0,1\}^n$	Boolean space of dimension n
$\vec{\mathbf{x}} = \begin{pmatrix} x_1 \\ \vdots \\ x_n \end{pmatrix}$	A column vector
$\vec{\mathbf{x}}^\top = (x_1 \cdots x_n)$	A row vector
$\|\| \vec{\mathbf{x}} \|\|$	L_2 norm of the vector $\mathbf{x}$
$\partial/\partial x f(x,y)$	Partial derivative of function $f(x,y)$ with respect to x
$\dot{x}$ or $\mathrm{d}x/\mathrm{d}t$	Total time derivative of x
$\frac{\mathrm{d}f(x)}{\mathrm{d}x}$	Total derivative of function $f(x)$ with respect to x
$\mathcal{X}$	Input or description space of the examples
$\mathcal{Y}$	Output or label space
$\mathcal{S}_L$	Learning set

$\mathcal{P}$	Set of positive training examples
$\mathcal{N}$	Set of negative training examples
$\mathcal{S}_T$	Test set
e	An example
$\vec{\mathbf{x}}_k$	An example description
$\vec{\mathbf{z}}_k = (\vec{\mathbf{x}}_k, y_k)$	A labeled example (description, class)
$y_k \in \mathcal{Y}$	The true label of an example provided by an "oracle"
$\mathbb{C}$	Space of possible target concepts
$c : \mathcal{X} \to \mathcal{Y}$	A target concept
$\mathcal{H}$	Hypothesis space
$h \in \mathcal{H}$	A hypothesis considered by the learner
$y = h(\vec{\mathbf{x}}) \in \mathcal{Y}$	Prediction of the hypothesis h about example $\vec{\mathbf{x}}$
Φ	A set of logical formulas
$\varphi \in \Phi$	A logical formula φ belonging to the set Φ
$\ell\big(c(\vec{\mathbf{x}}), h(\vec{\mathbf{x}})\big)$	The loss incurred when $h(\vec{\mathbf{x}})$ is predicted instead of the true label $c(\vec{\mathbf{x}})$
E	Energy
S	Entropy
$x_j{:}a$	Variable x_j is bound to the constant a
$\overline{x_j{:}a}$	Variable x_j is not bound to the constant a
m	Number of literals in a formula to be matched
n	Number of variables in a formula to be matched
N	Number of goods in each table in an example
L	Number of constants occurring in the tables of an example

1

Introduction

Learning involves vital functions at different levels of consciousness, starting with the recognition of sensory stimuli up to the acquisition of complex notions for sophisticated abstract reasoning. Even though learning escapes precise definition there is general agreement on Langley's idea (Langley, 1986) of learning as a set of "mechanisms through which intelligent agents improve their behavior over time", which seems reasonable once a sufficiently broad view of "agent" is taken. Machine learning has its roots in several disciplines, notably statistics, pattern recognition, the cognitive sciences, and control theory. Its main goal is to help humans in constructing programs that cannot be built up manually and programs that learn from experience. Another goal of machine learning is to provide computational models for human learning, thus supporting cognitive studies of learning.

Machine learning's roots

Among the large variety of tasks that constitute the body of machine learning, one has received attention from the beginning: the acquiring of knowledge for performing *classification*. From this perspective machine learning can be described roughly as the process of discovering regularities from a set of available data and extrapolating these regularities to new data.

Classification

Over the years, machine learning has been understood in different ways. At first it was considered mainly as an algorithmic process. One of the first approaches to automated learning was proposed by Gold in his "learning in the limit" paradigm (Gold, 1967). This type of learning provides an infinite sequence of pieces of data to the learner, who generates a model that *explains* the data. At each new input the learner updates its current model (the "hypothesis"), hoping, but never knowing for sure, that it is closer to the "correct" one.

Machine learning as an algorithm

Gold's paradigm

A fundamental change in machine learning was the recognition of its nature as a *search* problem (Mitchell, 1982). Given a set of data and some language(s) for describing the data and the target knowledge, learning consists in the exploration of a hypothesis space, guided by a heuristic, until a specified termination

Machine learning as search

condition is met; as the search space is usually too large to be explored exhaustively, the learner must have a criterion to evaluate and compare hypotheses. In order to facilitate the search the hypothesis space is usually internally structured according to a *generality* relation.

Just as learning is a fundamental task in any living organism, machine learning is a fundamental task in artificial intelligence as well. It is impossible to conceive a truly intelligent agent that is not provided with the ability to extend its knowledge and improve its performance over time.

Appealing as it may be, machine learning encounters severe difficulties, which even today hinder its full exploitation. The main obstacle to be overcome is that most machine learning algorithms are very demanding in terms of computational resources, especially those that are closer to the human process of learning. This concept of *computational complexity* in learning is the core around which this book is constructed.

Computational complexity of learning

For hundreds of years the abstract nature of mathematical truths required advances through proving theorems. Existence or constructive proofs were concerned with the logical soundness of the derived results, without any attention to their concrete attainability. The same was true for algorithms: the only relevant aspect was their correctness, not their practical execution. Mathematical knowledge appeared to scientists as only limited by human skill in discovering or inventing it.

With the advent of information science, things changed radically. In fact, logician Kurt Gödel's work provided clear evidence that the discovery of some mathematical truths may be intrinsically limited (Gödel, 1931). In fact, with his famous *incompleteness theorem* he proved that Hilbert's belief in the existence of an effective procedure determining the truth or falsity of any mathematical proposition was illfounded: thus the notion of *undecidability* was born. In order to understand this fundamental notion better we have to be more precise about the concept of an *algorithm*. The word "algorithm" derives from the name of the Persian mathematician Abu Abdullah Muhammad ibn Musa al-Khwarizmi, whose work introduced Arabic numerals and algebraic concepts to the western world. He worked in Baghdad in the ninth century, when the city was a centre of scientific studies. The ancient word *algorism* originally referred only to the rules of performing arithmetic using Arabic numerals but evolved via the Latin translation of al-Khwarizmi's name into *algorithm* by the 18th century. In its more intuitive formulation, an *algorithm* is a precise and unambiguous sequence of steps that, given a problem to be solved and some input data, provides the solution thereof.[1]

Gödel's incompleteness theorem

Algorithm

[1] Actually, one may clarify the difference between *procedures* and *algorithms* by reserving the latter name for procedures that terminate. As we are concerned only with the halting case, we will use the two terms interchangeably.

In general, a particular problem to be solved is a specific instance of a *class* of problems. For example, the problem of sorting in ascending order the elements of a vector $\vec{\mathbf{x}}$ of n integer numbers belongs to a class Π of similar problems containing all such vectors, each with a different length n and different content. The notion of decidability refers to the class of problems as a whole, not to a single instance. More precisely, given a class of problems Π, we will say that the class is *decidable* if there exists an algorithm that, given as an input any instance of the problem class, provides a solution. Then, undecidability does not prevent any single instance from being solved but, rather, it limits the *generality* of the algorithm for finding the solution in any instance. In other words, for a decidable class a single algorithm is able to solve any instance of the class whereas for an undecidable class every problem instance must be solved with, in principle, a different algorithm.[2]

Decidability

In order to prove that a problem class is undecidable one has to show that no unique algorithm solves all its instances. This is usually done by reducing the problem (class) to a known undecidable problem. A basic undecidable problem is the *halting* problem, proved undecidable by Alan Turing in 1936 (Turing, 1936). The halting problem consists of writing a general algorithm that, taking as input any algorithm $\mathcal{A}$ and some input data, outputs YES or NO, depending on whether $\mathcal{A}$ halts or continues ad infinitum. Clearly, given a specific algorithm it is usually possible, with more or less ease, to decide whether it will stop for any specific input. However, there is no general algorithm that is able to provide this decision for *any* input algorithm.

Halting problem

Even though undecidability may be interesting from a philosophical point of view, in that it might be considered as a limiting factor to human knowledge, this notion is not a subject of this book, in which we are concerned only with decidable problem classes.

But, even limiting the study to decidable problems, difficulties of another nature come up. These difficulties have been again brought to our attention in recent times by computer science, which stresses a concept that was not previously considered important in mathematics. As already mentioned, mathematical results are achieved by proving theorems or by designing abstract algorithms to solve problems. In computer science this is not sufficient: the algorithm for solving a problem must be *efficient*, i.e., it must run on a computer in *reasonable* time. In order to define what "reasonable" time means, the concept of the *computational complexity* of an algorithm must be introduced.

Efficient algorithms

Given an algorithm $\mathcal{A}$, working on some data, its computational complexity is related to its run time. However, the run time depends on the programming language used to implement the algorithm and on the specific machine on which

[2] In the following we will use, for the sake of simplicity and where no ambiguity may arise, the terms "class of problems" and "problem" interchangeably.

the program is run. In order to make its definition more precise, and independent of the specific implementation, the complexity is evaluated in terms of the number of elementary steps performed by the algorithm and not in terms of the time it takes to execute them. But, even so, there are uncertainties about what has to be considered an elementary "step" in an algorithm, because this depends on the granularity of the observation. To overcome this difficulty an ideal, abstract, computer model is used, for which the notion of a "step" is precisely defined.

Turing machine

There is more than one "ideal" computer, but one of the simplest and best known is the *Turing machine*, an abstract computational device introduced by Alan Turing in the late 1930s (Turing, 1936), long before the first actual computer was built. The simplest version of the Turing machine consists of a tape, a read−write head, and a control unit. The tape, infinite in both directions, is divided into squares which contain a "blank" symbol and at least one other symbol belonging to an alphabet Σ. A square numbered 0 separates the left and right parts of the tape. The head can read or write these symbols onto the tape. The control unit of the machine specifies a finite set of states in which the machine can be; at any point in time a Turing machine is in exactly one of these states. The control unit can be thought of as a *finite state automaton*. This automaton encodes the "program". The computation proceeds in steps: at each step the head reads the content of the square in which it is positioned and, according to this content and the current state of the automaton, it writes another symbol on the same square and then moves one square to the left or to the right. At the beginning the input data are written on the right-hand part of the tape, starting at position 0, and the rest of the tape is filled with "blanks". When the computation is over the machine stops, and the output can be read on the tape.

Notwithstanding the simplicity of its mechanism, the Turing machine is believed to be able to compute any computable algorithm that one can conceive. This assertion was hypothesized by Alonzo Church (1936) through the definition of the λ-*calculus* and the introduction of the notion of effective calculability: a function is said to be effectively calculable if its values can be found by some purely mechanical process. Later, Turing showed that his computation model (the Turing machine) and the λ-calculus are equivalent, so the assertion is now known as *Church−Turing thesis*. This thesis is almost universally accepted now, even though the extent of its applicability is still a subject of debate.

Church−Turing thesis

Implementing an algorithm on a Turing machine allows the notion of computational complexity to be precisely defined. A "step" in an algorithm is a cycle <read a symbol, write a symbol, move the head> and the execution of any program is a sequence of such steps. Given a (decidable) class Π of problems, let $\mathcal{T}$ be the particular Turing machine used to find the solution. If we take an instance $\mathcal{I}$ belonging to Π, let $\mathcal{C}_{\mathcal{T}}(\mathcal{I})$ be the exact number of steps $\mathcal{T}$ uses to solve $\mathcal{I}$. Clearly, however, $\mathcal{C}_{\mathcal{T}}(\mathcal{I})$ is too detailed a measure, impossible to evaluate before

the program is executed. Thus we need a less detailed measure; to this end let n be an integer characterizing the *size* of the problem at hand. For instance, coming back to the sorting problem, n can be the length of the vector to be sorted. We can partition the class Π into subclasses Π_n, each containing only instances $\mathcal{I}_n$ of length n. Even so, running the algorithm on the $\mathcal{I}_n$ requires different numbers of steps, depending on the specific instance. As we want a safe measure of complexity, we take the pessimistic approach of attributing to the subclass Π_n the highest complexity found in the $\mathcal{I}_n$, i.e., the complexity of the *worst case*. Formally, we define the complexity as

Formal definition of computational complexity

$$\mathcal{C}_{\mathcal{T}}(n) = \operatorname*{Max}_{\mathcal{I}_n \in \Pi_n} \{\mathcal{C}_{\mathcal{T}}(\mathcal{I}_n)\}.$$

As it can be proved that the complexity $\mathcal{C}_{\mathcal{T}}(n)$ is independent of the abstract Turing machine used and that it is the same for any concrete computer, we will drop the subscript $\mathcal{T}$ from the complexity and simply write

$$\mathcal{C}_{\mathcal{T}}(n) = \mathcal{C}(n).$$

The complexity $\mathcal{C}(n)$ defined above is called the *time complexity*, because it refers to the time resource consumption of the algorithm. In a similar way, the *space complexity* can be defined as the maximum amount of memory simultaneously occupied during the program's run. In this book, the word "complexity" will always refer to the time complexity unless otherwise specified.

The introduction of the subclasses Π_n is fundamental because the very goal of computational complexity theory is to estimate how the time complexity of an algorithm *scales up* with increasing n. In order to better understand this idea, let us consider two functions f and g from the integers to the integers:

$$f: \ \mathbb{N}^+ \to \mathbb{N}^+, \qquad g: \ \mathbb{N}^+ \to \mathbb{N}^+.$$

What we are interested in is the relative *order of magnitude* of the two functions rather than their actual values. So, let us consider the ratio of $f(n)$ and $g(n)$. There are three cases:

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = \infty; \tag{1.1}$$

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = a \neq 0, \infty; \tag{1.2}$$

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0. \tag{1.3}$$

In the first case $f(n)$ is of an order of magnitude greater than $g(n)$; in the second case it is of the same order; and in the third case it is of a lower order of magnitude. Introducing the $\mathcal{O}$ ("big O") notation, we will say that $f(n) = \mathcal{O}(g(n))$ in

the second and third cases, namely:

$$f(n) = \mathcal{O}(g(n)) \quad \leftrightarrow \quad \lim_{n\to\infty} \frac{f(n)}{g(n)} = a \neq \infty. \tag{1.4}$$

$\mathcal{O}$ notation

From definition (1.4) we deduce that in $f(n)$ and $g(n)$ all terms of lower orders of magnitude can be omitted, as well as multiplicative constants. It is worth noting explicitly that the $\mathcal{O}$ notation tells us only that $f(n)$ is of order of magnitude *not greater* than that of $g(n)$, not that $f(n)$ and $g(n)$ have the same order in the mathematical sense. If $f(n)$ and $g(n)$ have exactly the same order of magnitude (case (1.2)), we will say that $f(n) = \Theta(g(n))$. If $f(n)$ has order of magnitude strictly greater than $g(n)$, we will write $f(n) = \Omega(g(n))$.

—— EXAMPLE ——

We give here some examples of the $\mathcal{O}$ notation:

$$\begin{aligned} f(n) &= 50n^2 + 20n + 3 = \mathcal{O}(n^2) \\ f(n) &= n^3 + 30n^2 + 100 = \mathcal{O}(n^3) \\ f(n) &= 3^n + 100^3 = \mathcal{O}(3^n) \end{aligned}$$

Given two functions $f(n)$ and $g(n)$, we can provide a formal definition of $f(n) = \mathcal{O}(g(n))$ as follows:

$$f(n) = \mathcal{O}(g(n)) \quad \leftrightarrow \quad \exists n_0 \; \exists c \; [\forall n > n_0 : f(n) \leqslant cg(n)], \tag{1.5}$$

where c is a positive constant and n_0 is a positive integer. Definition (1.5) tells that what happens for low values of n (i.e., lower than n_0) is not important; on the contrary, only the asymptotic behavior of $f(n)$ is relevant. A graphical illustration of definition (1.5) is given in Figure 1.1. We are now in a position to define precisely what a reasonable complexity is. An algorithm $\mathcal{A}$ will be said to be *efficient* if it runs with a complexity that is at most *polynomial* in the size n of the input. Actually, many problems that are relevant in practice show a much more rapid (exponential) increase in the time required to obtain a solution, when the size of the problem increases; such problems cannot be solved within acceptable time spans.

Polynomial complexity

In computer science the study of the computational complexity of algorithms takes a central place; since its beginnings, scientists have studied problems from the complexity point of view, categorizing their behavior into a well-known *complexity class hierarchy*. According to the needs of this book we show, in Figure 1.2, an oversimplified version of this hierarchy, including only three types of problem: $\mathbf{P}$, $\mathbf{NP}$, and $\mathbf{NP}$-complete. The class $\mathbf{P}$ contains problems that can

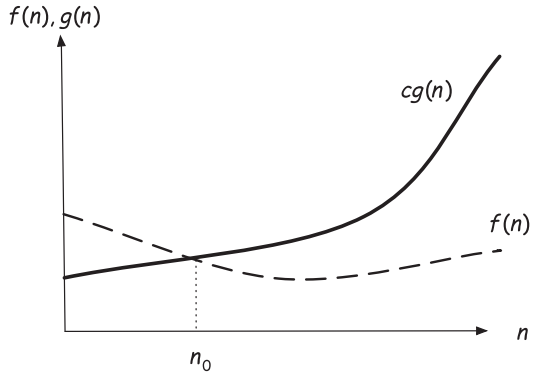


Figure 1.1 Graphical illustration of the $\mathcal{O}$ notation. If $f(n) = \mathcal{O}(g(h))$ then after a given threshold n_0, the function $f(n)$ must be smaller than $cg(n)$, where c is a positive constant. What happens for $n < n_0$ does not matter.

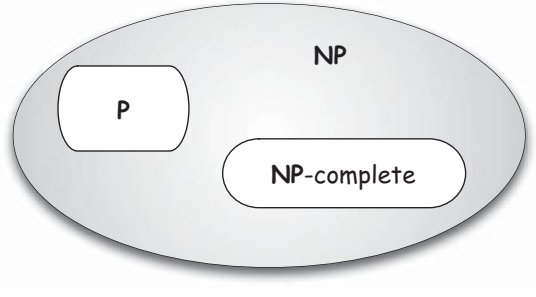


Figure 1.2 Simplified version of the complexity class hierarchy. The class $\mathbf{NP}$ includes both the class $\mathbf{P}$ and the class of $\mathbf{NP}$-complete problems.

be solved in *polynomial* time by a Turing machine like the one described above, i.e., a *deterministic* Turing machine. In order to define the class $\mathbf{NP}$ the behavior of a deterministic Turing machine must be extended with a nondeterministic phase, to be executed at the beginning, thus becoming a *non-deterministic* Turing machine. In the non-deterministic phase, a potential solution is generated; this is then verified by the subsequent deterministic phase. The $\mathbf{NP}$ class contains those problems that can be solved in *polynomial* time by such a *non-deterministic* machine. Whereas the deterministic Turing machine captures the notion of the polynomial *solvability* of a problem class, the non-deterministic Turing machine captures the notion of the polynomial *verifiability* of a problem

class. In other words, if one is able to suggest a potential solution to a problem, the non-deterministic Turing machine can verify in polynomial time whether it is indeed a solution. It should be clear from the definition that $\mathbf{P}$ is a subclass of $\mathbf{NP}$: polynomial solvability implies polynomial verifiability.

$\mathbf{P} = \mathbf{NP}$?

Today, the question whether $\mathbf{P} = \mathbf{NP}$ is still an important open problem in computer science. On the one hand it has not been possible to prove that $\mathbf{P} = \mathbf{NP}$ and on the other hand no polynomial algorithm has been found for many problems in $\mathbf{NP}$, notwithstanding the amount of effort devoted to the task. Thus, the general opinion is that $\mathbf{P} \neq \mathbf{NP}$. This opinion is strongly supported by the existence of the subclass $\mathbf{NP}$-complete. This subclass contains (a large number of) problems that share the following property: each problem in $\mathbf{NP}$ (including $\mathbf{P}$) can be reduced in polynomial time to any problem in $\mathbf{NP}$-complete; as a consequence, it would be sufficient to solve in polynomial time just one problem in $\mathbf{NP}$-complete to prove that $\mathbf{P} = \mathbf{NP}$. Given the amount of work already devoted to this task, it seems highly unlikely that this will turn out to be the case. In this book we will assume as an underlying hypothesis that $\mathbf{P} \neq \mathbf{NP}$.

Combinatorial problems

A class of problems that are particularly prone to a dramatic increase in computational complexity with increasing problem size is the class of *combinatorial* problems, many among which are $\mathbf{NP}$-complete. Informally, a combinatorial problem is one that requires combinations of objects belonging to a set to be explored, with the goal of deciding whether a specified property holds true or of finding some optimal combination according to a specified criterion. Combinatorial problems are well represented in artificial intelligence, operational research, complex system analysis, and optimization and search. Among the large variety of existing combinatorial problems, two have received a great deal of attention, namely the *satisfiability* (SAT) problem (Cook, 1971) and the *constraint satisfaction problem* (CSP) (see for instance Kumar, 1992). In Chapters 3 and 4 these two problems will be introduced and described in detail, because they are intimately related to machine learning and to the sources of its complexity.

Typical complexity

Reconsidering the definition of computational complexity provided earlier, it is not necessarily a good idea to take the worst-case complexity as that representative of an algorithm $\mathcal{A}$. In fact, $\mathcal{A}$ might be able to provide a solution in reasonable time for the majority of the instances in Π_n, running for a very long time in only a few particular instances; this is the case, for example, for the branch-and-bound optimization algorithm (Lawler and Wood, 1966). For this reason a new paradigm has emerged, which uses the *typical* running behavior of an algorithm instead of its worst case. The notion of the typical complexity has a precise meaning. Namely, it requires two conditions:

- The typical complexity is the most probable complexity over the class of problem instances considered.

- As each problem instance has its own run time, there is a difference in complexity between that for the specific instance and the most probable complexity. When the size of the considered instances grows to infinity, the difference between their complexity and the most probable complexity must go to 0 with probability 1.

This new perspective on the complexity of algorithms was suggested by the discovery of interesting and fruitful links (previously unsuspected) between combinatorial problems and systems obeying the laws of statistical physics. It turns out that combinatorial problems share several characteristics with physical systems composed of a large number of particles and that a precise parallel can be established between such physical entities on the one hand and combinatorial functions to be optimized on the other hand.

Phase transitions

Among the many parallels that can be drawn from the link between such many-body physical systems and computational systems, one aspect is particularly relevant for this book, namely, the emergence of a *phase transition* (see for instance Hogg, 1996). Some physical systems composed of a large number of particles may exist in different *phases*. A phase is a homogeneous (with respect to some specified physical quantity) state of the system. A well-known case in everyday life is water, which can exist in solid, liquid, and gaseous phases. The phase that water is in depends on the values of the macroscopic variables describing the physical state, for instance, the temperature and pressure. In Figure 1.3 a qualitative schema of the phases in which water may exist is shown. In a phase transition we distinguish between the order and control parameters: an *order* parameter is a quantity that shows a marked difference in behavior across the transition line whereas a *control* parameter is one that determines the location of the transition. In the case of water, a possible order parameter is the density whereas a possible control parameter is the temperature. The order and control parameters characterize the phase transition.

Order and control parameters

Types of phase transitions

According to Ehrenfest's classification, there are two types of phase transition, namely *first-order* and *second-order*. A precise definition of these types will be given in Chapter 2. We just mention, here, that we are interested in first-order phase transitions; in this type of transition, in addition to the discontinuity in the order parameters, there is usually another quantity that goes to infinity when the size of the system tends to infinity as well. Moreover, at the transition point the two phases coexist. In the case of water, for example, the specific heat diverges at the transition between liquid and vapor, because heat is being supplied to the system but the temperature remains constant.

In computational problems the order parameters are usually quantities that characterize aspects of the algorithm's behavior (for instance, the probability that a solution exists) and the control parameters describe the internal structure

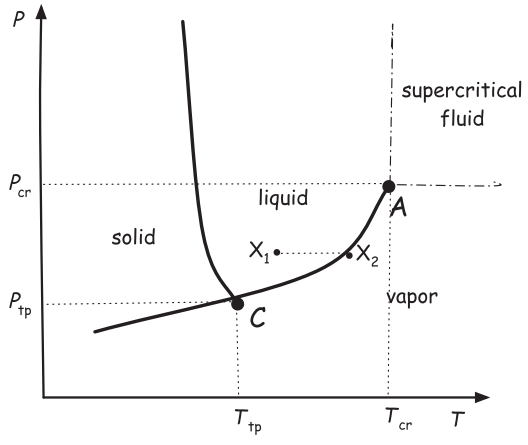


Figure 1.3 Qualitative diagrams of the phases in which water can exist. The temperature T and pressure P are the control parameters of the transition between phases. Along the separation lines two of the phases coexist. At the triple point C all three phases are present. Beyond the critical point A the water is said to be in a supercritical fluid state. In this state the molecules are too energetic and too close to each other for a clear transition between liquid and vapor to exist.

of the problem, whereas the quantity that diverges at a "phase transition" is the computational complexity of the algorithm.

There are various motivations for studying the emergence of phase transitions. First, their emergence seems to be an ubiquitous phenomenon in many-body systems, capturing some essential properties of their nature. They occur not only in physical and computational systems but also in human perception and social sciences, as will be described later in this book. Second, systems that show a phase transition exhibit, at the transition point, interesting singularities in behavior called "critical phenomena", which elucidate their real essence, in a way not evident by other means. Third, phase transitions are interesting in themselves, as they explain ensemble or macroscopic behaviors in terms of short-range microscopic interactions.

For computational systems the discovery of a phase transition in a problem class has several important consequences. The phase transition region contains the most difficult problem instances, those for which the computational complexity shows an exponential increase with the problem size. Also, the phase transition can be used as a source of "difficult" test problems for assessing the properties and the power of algorithms and for comparing them in meaningful problem instances. Moreover, very small variations in the control parameter

value may induce very large variations in the algorithm's behavior and/or in the types of solution. Thus, a knowledge of the critical value of a control parameter allows the user to roughly predict the behavior of the algorithm. Moreover, by exploiting further the analogy with physical systems, it is possible to highlight the very sources of the complexity, which is not possible within classical complexity theory. In fact, with the techniques derived from statistical physics it is possible to enter into the deep structure of the problem and of its solutions; the system's behavior near the phase transition allows a microscopic view of the solution space of the problems to be investigated by exploiting the correspondence established by the link. This fact not only offers the possibility of a deeper understanding of the properties of algorithms but also opens the way to the introduction of effective new algorithms.

Even though the emergence of phase transitions is a widely diffused phenomenon in many fields, we will limit ourselves in the main to machine learning. We will then consider different approaches to machine learning and show how the emergence of phase transitions affects their feasibility in a radical way.

2

Statistical physics and phase transitions

Contents

2.1 Basic notions of statistical physics

In order to make this book self-contained, some basic notions of statistical physics will be introduced in this chapter. In the main we have followed the approach of Landau and Lifshitz, to which the interested reader is referred, if he or she wants to go deeper into the subject (Landau and Lifshitz, 1976, 1980).

In dynamics (Landau and Lifshitz, 1976), a central role is played by the notion of a *point particle*, which is a body with a finite mass m whose size can be neglected when describing its motion, so that it can be geometrically assimilated to a point. The position of a point particle is given by a vector $\vec{\mathbf{r}}$ in the Cartesian

Point particle

coordinate space (x, y, z), and its velocity is the time derivative of $\vec{\mathbf{r}}$:

$$\vec{\mathbf{v}} = \frac{\mathrm{d}\vec{\mathbf{r}}}{\mathrm{d}t}.$$

Degrees of freedom

Let $\mathcal{S}$ be a closed dynamical system composed of N particles. In order to determine the configuration of the system in Cartesian space, N vectors must be given, one for each component point. The number of independent values that together determine the system's position is its number of *degrees of freedom*, which in this case is $3N$ because each vector has three components (x, y, z).

Generalized coordinates

As Cartesian coordinates may not always be the best choice for describing the motion of a system, a set of s *generalized coordinates* $\{q_1, q_2, \ldots, q_s\}$ is used instead. Generalized coordinates are any set of variables able to characterize precisely the position of a system at a given time. The number s is the number of degrees of freedom. However, knowledge of the coordinates q_i $(1 \leqslant i \leqslant s)$ is not sufficient to predict the future configurations of the system $\mathcal{S}$; for this, the *generalized velocities* $\dot{q}_i$ $(1 \leqslant i \leqslant s)$ as functions of time must be known.[1] This allows the *accelerations* of the system to be obtained; then, providing the values of the generalized coordinates and velocities at a given time determines the motions in the system for all future times.

—— EXAMPLE ——

In order to clarify the notion of generalized coordinates, let us consider the system shown in Figure 2.1. Clearly the configuration of the whole system is completely determined if we know the value of x_A (the position of A on the x axis) and the angle θ of the rod with respect to the vertical axis y. The values x_A range in $(-\infty, \infty)$ whereas θ ranges in the interval [0, 2π). We can define two generalized coordinates, $q_1 = x_A$ and $q_2 = \theta$. The relation between the Cartesian coordinates of A and B and the generalized coordinates is as follows:

$$\begin{aligned} x_A &= q_1, & y_A &= 0, \\ x_B &= q_1 + \ell \sin q_2, & y_B &= \ell \cos q_2. \end{aligned}$$

Analogous relationships hold between the time derivatives:

$$\begin{aligned} \dot{x}_A &= \dot{q}_1, & \dot{y}_A &= 0, \\ \dot{x}_B &= \dot{q}_1 + (\ell \cos q_2)\dot{q}_2, & \dot{y}_B &= (-\ell \sin q_2)\dot{q}_2. \end{aligned}$$

[1] For simplicity we use q to denote the set $\{q_1, q_2, \ldots, q_s\}$ and $\dot{q}$ to denote the set $\{\dot{q}_1, \dot{q}_2, \ldots, \dot{q}_s\}$.

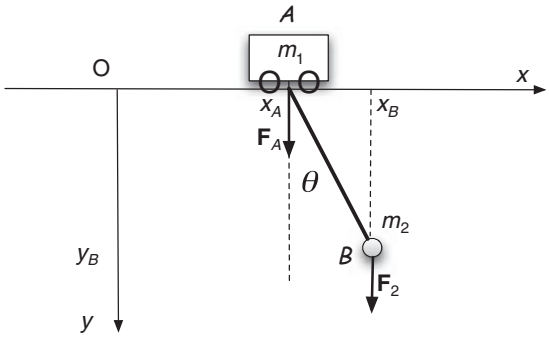


Figure 2.1 A small body A of mass m_1 moves along the horizontal x-axis. A point particle B of mass m_2 is attached to A by a rigid rod of length ℓ and negligible mass, hinged at A.

Lagrangian function

The relationships linking the accelerations to the positions and velocities are called *equations of motion* and can be obtained using the *Lagrangian function* $\mathcal{L}(q, \dot{q}, t)$ of the system, which, if the system is only subject to conservative fields, can be written as

$$\mathcal{L}(q, \dot{q}, t) = \frac{1}{2} \sum_{i=1}^{s} \sum_{j=1}^{s} a_{i,j}(q) \dot{q}_i \dot{q}_j - U(q). \tag{2.1}$$

Kinetic energy

In (2.1) the first summation is quadratic in the velocity components and represents the *kinetic energy* $K(q, \dot{q})$ of the system. If Cartesian coordinates are used instead of generalized coordinates, this term becomes the better-known expression

$$K(\vec{\mathbf{v}}) = \frac{1}{2} \sum_{k=1}^{N} m_k |\vec{\mathbf{v}_\mathbf{k}}|^2,$$

where $|\vec{\mathbf{v}_\mathbf{k}}|$ is the modulus of the velocity of particle p_k.

The second term on the right-hand side of (2.1) captures the *potential energy* of the system, which (for conservative fields) is a function of position only. Using the Lagrangian function, the equations of motion for the system are as follows:

$$\frac{\mathrm{d}}{\mathrm{d}t} \frac{\partial}{\partial \dot{q}_i} \mathcal{L} - \frac{\partial}{\partial q_i} \mathcal{L} = 0 \qquad (1 \leqslant i \leqslant s). \tag{2.2}$$

Lagrange equations

The *Euler–Lagrange equations* (2.2) are a set of second-order differential equations whose general integral contains $2s$ constants. These constants can be determined by providing the values of the coordinates and velocities at a given time,

conventionally $t_0 = 0$. In addition we can define a coordinate conjugate to $\dot{q}_i$:

$$p_i = \frac{\partial}{\partial \dot{q}_i}\mathcal{L}.$$

The quantity p_i is called a *generalized momentum*. In Cartesian coordinates the momentum of a point particle with respect to the ith coordinate, is simply Generalized momentum

$$p_i = m_i v_i.$$

Another fundamental function describing the dynamics of a system is the *Hamiltonian* H, which represents the total energy. The Hamiltonian can be expressed as follows: Hamiltonian

$$H(q, p, t) = \sum_{i=1}^{s} p_i \dot{q}_i - \mathcal{L}, \tag{2.3}$$

which, using (2.1), becomes

$$H(q, \dot{q}, t) = K(q, \dot{q}) + U(q). \tag{2.4}$$

Moreover, using equations (2.3) it is possible to see that

$$\frac{\partial}{\partial t} H = 0$$

if $\partial L/\partial t = 0$. In other word, H does not depend explicitly upon time if L does not. By integrating the equations of motion (2.2) for a closed system,[2] it can be seen that even though the coordinates and the velocities change with time there are certain quantities, called the *first integrals* of motion, which remain constant in time. The one in which we are mostly interested in this book is the *energy*, which is the value assumed by the Hamiltonian function for a particular set of coordinate values. Energy

EXAMPLE

Let us consider again the system depicted in Figure 2.1. Now suppose that the objects A and B, with masses m_1 and m_2, are subject to a conservative gravitational field, which applies forces $|\vec{\mathbf{F}}_1| = m_1 g$ and $|\vec{\mathbf{F}}_2| = m_2 g$ to A and B, respectively. Moreover, A has a translational motion along the x-axis whereas B has a composite motion, a translation along x and a rotation around A. Using the two generalized coordinates $q_1 = x_A$ and $q_2 = \theta$, we can write the Lagrangian of the system as follows:

$$\mathcal{L}(q, \dot{q}, t) = \left[\frac{1}{2}(m_1 + m_2)\dot{q}_1^2 + \frac{1}{2} m_2 \ell^2 \dot{q}_2^2\right] + m_2 g \ell \cos q_2.$$

[2] By definition, a closed system is one that does not exchange energy or matter with any body not contained in the system itself.

The last term is the potential energy $U(q)$. For a gravitational field we have

$$U(q) = -m_1 g y_A - m_2 g y_B = -m_2 g \ell \cos q_2 .$$

In order to obtain the above expression for the potential energy, we set the level of zero potential energy at $y = 0$. Then the first term in the middle expression disappears, and only the potential energy of B remains. (Notice that the y-axis points downward and so the minus sign is required in order that $U(q)$ decreases from top to bottom.)

The generalized momenta are then

$$p_1 = \frac{\partial}{\partial \dot{q}_1}\mathcal{L} = (m_1 + m_2)\dot{q}_1 , \qquad p_2 = \frac{\partial}{\partial \dot{q}_2}\mathcal{L} = m_2 \ell^2 \dot{q}_2 .$$

We can now write down an expression for the Hamiltonian:

$$\begin{aligned} H(q,p,t) &= p_1\, \dot{q}_1 + p_2\, \dot{q}_2 - \mathcal{L}(q,\dot{q},t) \\ &= \left[\frac{1}{2}(m_1 + m_2)\dot{q}_1^2 + \frac{1}{2} m_2 \ell^2 \dot{q}_2^2\right] - m_2 g \ell \cos q_2 . \end{aligned}$$

Finally, the two equations of motion are derived from (2.2):

$$\begin{aligned} (m_1 + m_2)\ddot{q}_i &= 0 \quad \rightarrow \quad \ddot{q}_i = 0, \\ m_2 \ell^2 \ddot{q}_2 + m_2 g \ell \sin q_2 &= 0 \quad \rightarrow \quad \ddot{q}_2 = -\frac{g}{\ell}\sin q_2 . \end{aligned}$$

If the system $\mathcal{S}$ is composed of only a small number of particles, the equations of motion (2.2) can be solved, at least numerically (an exact solution in closed form is known only for $N < 3$). But, when the number of particles becomes very large (of the order of the *Avogadro number* $N_A = 6.022 \times 10^{23}$ mol^{-1}), it is impossible to do so. Then the microscopic behavior of the system cannot be precisely captured, not even in numerical form. One might think, then, that the behavior of such a system must be so complex that it is impossible to describe.

Statistical laws

Actually there is a way out of this situation, because the sheer number of microscopic components, which hinders a deterministic solution of the equations of motion, lets laws of a different nature emerge, namely, *statistical* laws. These laws, even though not reducible to purely mechanical ones, allow the value and behavior of macroscopic quantities of the system to be described and predicted. The study of these laws is the subject of *statistical mechanics* or, more generally, *statistical physics*, in which the macroscopic behaviors of systems composed of a large number of particles are explained in terms of *statistical ensembles*.

Phase space

In order to understand how this can be done, let us introduce the notion of *phase space*. The phase space of a system $\mathcal{S}$ is a $2s$-dimensional space whose

axes correspond to the s generalized coordinates q_i and the s generalized momenta p_i (rather than the generalized velocities). Each point in the phase space is associated with a *state* of the system $\mathcal{S}$. As time progresses, the point representing the state of the system moves around in the phase space and thus its trajectory describes the time evolution of $\mathcal{S}$.

The fundamental observation underlying statistical physics is that, owing to the enormous complexity of the trajectories of the component particles, during a sufficiently long time interval T the system will find itself a large number of times in every possible state. More precisely, let $\Delta q \Delta p$ be a small volume in the phase space corresponding to values of the q's and the p's situated in small intervals Δq and Δp with their origin in (q, p). Let Δt be the amount of time the system spends inside the volume $\Delta q \Delta p$ during T. Then, as $T \to \infty$, the ratio $\Delta t/T$ becomes the probability that system $\mathcal{S}$ can be found, at a random instant, in the volume $\Delta q \Delta p$:

$$\mathtt{P}(\mathcal{S} \text{ is in } \Delta q \Delta p) = \lim_{T \to \infty} \frac{\Delta t}{T}. \tag{2.5}$$

Statistical distribution

Equation (2.5) was derived by following the system $\mathcal{S}$ in its evolution in time. More precisely, (2.5) is a consequence of the *principle of equiprobability* of the states reachable by a system with energy E.

However, if we look at the volume $\Delta q \Delta p$ in the long term and accumulate the transits of the system into and out of that volume, we may consider each transit as a stationary copy of the system and rewrite the probability (2.5) in terms of the "density" of occupied states within this volume. By letting the volume become infinitesimal, so that the probability becomes infinitesimal as well, we can write:

$$d\mathtt{P} = \rho(q, p) dq dp \tag{2.6}$$

The function $\rho(q, p)$ is the probability density in the phase space and is called the *statistical distribution function* of the system under consideration. The function $\rho(q, p)$ is normalized over the whole phase space. An interesting property of this distribution function is that it does not depend on the initial state of the system, provided that sufficient time has passed to allow the system to transit multiple times to each possible state.

Phase space average

The equivalence between the evolution in time of the single system $\mathcal{S}$ in phase space and the stationary distribution of the state density of "copies" of the same system has important consequences. Let $f(q, p)$ be a physical quantity depending on q and p and hence implicitly on the time t. On the one hand, the mean value $\bar{f}$ of $f(q, p)$ can be computed as follows:

$$\bar{f} = \int_{\Omega} f(q, p) \rho(q, p) dq dp, \tag{2.7}$$

Temporal average

where the integral is taken over the whole phase space Ω. Averaging in this phase space avoids the necessity of following the evolution in time of the function $f(t) = f(q(t), p(t))$. On the other hand, equation (2.5) tells that the statistical mean (2.7) is equivalent to the temporal mean and hence

$$\bar{f} = \lim_{T\to\infty} \frac{1}{T} \int_0^T f(t)dt. \tag{2.8}$$

Ergodicity

Systems for which this equivalence holds are called *ergodic*. Ergodicity is still a debated open problem in physics; its theoretical validity has been questioned but it is widely assumed to hold in practice because such an assumption gives good predictions.

Statistical mechanics makes it possible to predict the behavior of macroscopic bodies composed of many particles.

Equations (2.7) and (2.8) show that in statistical mechanics it is possible to make predictions, stochastic in nature, about the behavior of macroscopic bodies composed of many particles. This is the main difference from classical mechanics, which produces deterministic, exact, predictions only for one- and two-body systems. We note that the stochastic character of the results from statistical physics is not inherent in the nature of the systems considered (as it is in quantum mechanics) but derives from the fact that such results are obtained from an amount of data much smaller than that necessary to obtain a precise and complete mechanical description.

However, when the statistical approach is applied to macroscopic bodies the stochastic character of the predictions is not apparent. In fact, after a long time, all physical quantities describing an isolated macroscopic body become constant and equal to their mean value. In other words, fluctuations around the mean value are then negligible because the probability distribution for any quantity f will be strongly peaked around the mean. If this is the case, the body is said to be in *thermodynamic equilibrium* (or *statistical* equilibrium). The time necessary for an isolated system to reach thermodynamic equilibrium is the *relaxation time*. To understand this, let us consider a quantity f characterizing a

Fluctuations

macroscopic body and let $\bar{f}$ be its mean value. The values of f oscillate around $\bar{f}$, generating an instantaneous *difference* $\Delta f = f - \bar{f}$. The mean value $\overline{\Delta f}$ is not a good measure of fluctuation, however, because it will be zero, being average of values that are sometimes positive and sometimes negative. A better measure is the square root of the *quadratic difference*

$$\sqrt{\overline{(\Delta f)^2}}.$$

If f is *additive*, i.e., its value for a whole system is the sum of the values for the component parts, then an important consequence follows, namely

$$\frac{\sqrt{\overline{(\Delta f)^2}}}{\bar{f}} \sim \frac{1}{\sqrt{N}}, \tag{2.9}$$

where N is the number of components of the system. From (2.9) it is easy to see why fluctuations are not observed in a macroscopic body: their amplitude decreases with $\sqrt{N}$ and hence tends to zero rapidly.

Another fundamental quantity in statistical physics is the *entropy* S. The entropy of a system is defined in terms of the number of states in which it can be. For a system with Ω distinct microscopic states, we have

Entropy

$$S = -k_B \ln \Omega. \tag{2.10}$$

Here $k_B = 8.617\,343 \times 10^{-5}$ eV/°K is the *Boltzman constant*. The value $\Omega(E, V, N)$, which is a mechanical quantity, is a function of the energy E, the volume V, and the number of particles N of the system. The entropy of a system, which is a thermodynamic quantity, is linked to its *disorder*. Low values of entropy correspond to more ordered states (the probability of finding the system in a given state is concentrated in only a few states), whereas high values of entropy correspond to less ordered states (the probability of finding the system in a given state is distributed among many states).

Boltzman constant

If external forces are applied to a system, they can produce *work*. According to mechanics, the work done is the sum of the scalar products of the forces and the corresponding displacements. Doing work on a body may set it in motion or change its volume. The work performed on a body (or received from it) can be expressed as the change in a new thermodynamic quantity, the *free energy*, which is a function of the state of the body. As the work given or obtained depends on the thermodynamical conditions of the process, the definition of the free energy is not unique.

Free energy

2.2 Ensemble of states

In order to make predictions and to compute interesting properties of many-body systems, it is necessary to introduce the notion of an *ensemble* of states. An ensemble is specified by a set of allowed states for the system and an associated probability distribution over the states. Three types of ensemble are used: the canonical, microcanonical, and grand canonical ensembles.

2.2.1 Microcanonical ensemble

In an N-body system we call a *microstate* a complete description of every particle in the system. In other words, each point in the system's phase space corresponds to a *configuration*, i.e., for systems described by classical mechanics, a complete specification of the position and velocity of each particle. In quantum mechanics, however, a microstate or configuration would be specified by

Microstate

Configurations

the complete many-particle wavefunction. There is thus a one-to-one correspondence between points in the phase space and configurations. In the following we will consider only *discrete* configurations. Let $\Gamma = \{\gamma_1, \gamma_2, \ldots, \gamma_n\}$ be the set of such configurations. For an isolated system, in which neither energy nor matter is exchanged with the surroundings, Γ constitutes a *microcanonical* ensemble. Thus a microcanonical ensemble has constant energy.

It is impossible to compute the microstates of a large system exactly. Therefore, different microstates that lead to the same values of macroscopic properties such as the volume, temperature or energy are associated with a unique macrostate. It is then possible to imagine many copies of the system such that each copy is in a different configuration but is described by the same macrostate.

Macrostate

In order to derive the macroscopic properties of the system, a probability distribution must be associated with the microcanonical ensemble, i.e., with each configuration (microstate). Thus, the thermodynamic properties of the system can be computed as averages over all copies in the ensemble. The fundamental assumption of thermodynamics is that each configuration (microstate) occurs with the same probability.

More precisely, in the microcanonical ensemble all copies of the system have the same number N of particles, the same volume V, and the same energy E. If Ω is the number of microstates in the ensemble, the probability that a copy of the system, chosen at random, would be in a given configuration γ_i is simply

$$\mathrm{P}(\gamma_i) = \frac{1}{\Omega}. \tag{2.11}$$

For some types of system, for instance for an ideal gas, it is possible to compute Ω in an approximate way. In a microcanonical ensemble the entropy S is given by expression (2.10).

Consideration of (2.11) suggests that, for the microcanonical ensemble, Ω assumes the role played by the partition function in a canonical ensemble. For this reason, it is sometimes called the microcanonical partition function.

2.2.2 Canonical ensemble and Gibbs distribution

In a *canonical* ensemble the number N of particles, the system volume V, and the temperature T are constant, whereas there are no constraints on the particle momenta. In all states of the ensemble, all N particles lie within the volume V. The shape of the volume is unspecified, and it is not important as long as the volume is sufficiently large for surface phenomena to be ignored. The thermodynamic properties do not depend on the shape of V. If a system is in contact with a *thermal bath* (i.e., a body with very large thermal inertia, so that its temperature can be considered constant), the constant temperature T of the system does not

Canonical ensemble

affect the set of its allowed states, but enters into the calculations only through the probability of the states.

The canonical ensemble describes such a closed system, in which matter cannot be exchanged with the environment, but energy is exchanged between the system and the thermal bath to keep the system's temperature constant. The most useful thermodynamic function in this case is the Helmholtz free energy F (see (2.18)). The microstates s_i in a canonical ensemble have probabilities given by the *Gibbs distribution*:

Gibbs distribution

$$\mathtt{P}(s_i) = \frac{1}{Z}\, e^{-H(s_i)/k_B T} = \frac{1}{Z}\, e^{-\beta H(s_i)}, \tag{2.12}$$

where $H(s_i)$ is the Hamiltonian of the system in microstate s_i, k_B is Boltzmann's constant, and T is the absolute temperature. Gibbs' law (2.12) asserts that the probability of occurrence of a microstate depends only on its energy. The parameter $\beta = 1/k_B T$ is known as the *inverse temperature*. The normalization constant Z is the *partition function*, whose value can be computed from

$$Z = \sum_{i=1}^{n} e^{-H(s_i)/k_B T} = \sum_{i=1}^{n} e^{-\beta H(s_i)}, \tag{2.13}$$

The symbol Z for the partition function probably derives from the term *Zustatensummen*, used by Boltzmann

where n is the number of states in the ensemble.

For the distribution (2.12) there are two limiting cases:

- *Infinite temperature*, $T \to \infty$ In this case all the configurations become equiprobable and the system is totally "disordered", i.e., it does not show any internal structure (like a gas).

- *Zero temperature*, $T \to 0$ The probability is concentrated around the configuration with minimum energy, called the *ground state*. Classically, the particles assume precise positions, such that the total force on them vanishes, and the system is in a totally ordered state (like a crystal).

Partition function

Starting from formula (2.13), the partition function Z can be written in a different way. The Hamiltonian function of the system assumes, in the n states, a finite set of m distinct values E_k; in fact, several states may have the same energy. Then, denoting by $M(E_k)$ the number of states with energy E_k, we can write (2.13) as follows:

$$Z = \sum_{k=1}^{m} M(E_k) e^{-\beta E_k}. \tag{2.14}$$

The partition function, beyond its role as a normalization factor, has deep links with the thermodynamic quantities introduced in the previous subsection. First,

we consider the energy E. The mean value of E with respect to the Gibbs distribution, denoted by $\langle E \rangle$, can be computed as follows:

$$\langle E \rangle = \sum_{k=1}^{m} E_k M(E_k) e^{-\beta E_k}. \tag{2.15}$$

Notice that the average energy $\langle E \rangle$ depends on the temperature T, as does Z. Let us now consider the natural logarithm of Z, namely $\ln Z$, and take its derivative w.r.t. β:

$$-\frac{\partial}{\partial \beta} \ln Z = -\frac{1}{Z} \sum_{k=1}^{m} M(E_k) e^{-\beta E_k} (-E_k) = -\frac{1}{Z} Z \langle E \rangle = \langle E \rangle. \tag{2.16}$$

ln Z as a generating function

Analogously, the variance of E with respect to the Gibbs distribution is the second derivative of $\ln Z$ with respect to β. More generally, $\ln Z$ is the *generating function* of the coefficient $M(E_k)$.

Let us now go back to the notions of entropy and free energy. By using the Gibbs distribution and definition (2.10) we can compute the entropy S as follows:

$$S = \frac{\langle E \rangle}{T} + k_B \ln Z.$$

The second term on the right-hand side is related to the *Helmoltz free energy* F by

$$F = -k_B T \ln Z, \tag{2.17}$$

and hence

$$F = \langle E \rangle - TS. \tag{2.18}$$

In a canonical ensemble a system tends to minimize its free energy. From a thermodynamics point of view the entropy is minimal when the free energy is minimal. As a consequence, according to (2.18) there is a competition between the entropy and the mean energy. In fact, minimizing F implies (i) minimizing the mean energy, which, in turns, decreases the entropy, and (ii) maximizing the entropy (owing to the minus sign), which, in turn, increases the mean energy. This trade-off between mean energy and entropy leads to a thermodynamic equilibrium. It may be noted that the mean energy is dominant at low temperatures, whereas the entropy is dominant at high temperatures.

2.2.3 Grand canonical ensemble

The *grand canonical* ensemble is an extension of the canonical ensemble, in the sense that a system described by a grand canonical ensemble is in equilibrium with a thermal bath with which it can exchange both particles and energy; it is

said to be an open system. A grand canonical ensemble is useful when the number of particles in a system cannot be determined easily: as this number is not known, the partition function of the grand canonical ensemble is evaluated as a weighted sum of partition functions of canonical ensembles with varying numbers of particles. The relevant thermodynamic function is the thermodynamic potential Ω, also called the *Landau potential*.[3]

Partition function for grand canonical ensemble

By denoting as $Z(N, V, T)$ the partition function of a canonical ensemble with the same V and T as a grand canonical ensemble and with N particles, the partition function of the grand canonical ensemble is defined as follows:

$$\mathcal{Z}(z, V, T) = \sum_{N=0}^{\infty} z^N \, Z(N, V, T) = \sum_{N=0}^{\infty} \sum_{i=1}^{n} z^N \, e^{-H(s_i)/k_B T}. \quad (2.19)$$

In equation (2.19), n is the number of microstates s_i and the parameter z is the *fugacity*, which represents the ease with which a new particle may be added to the system.

2.3 Phase transitions

Systems whose behavior is governed by statistical physics laws frequently show emergent phenomena, absent in classical mechanics, such as the appearance of a *phase transition*. According to Martin *et al.* (2001), "From a statistical mechanics perspective, a phase transition is nothing but the onset of non-trivial macroscopic (collective) behavior in a system composed of a large number of elements that follow simple microscopic laws". More simply, Lee and Yang (1952) defined a phase transition as *a singularity in the partition function* of a system (Blythe and Evans, 2003). If a system has a phase transition then it can be in one of several phases, depending on the values of certain control parameters. Each phase is characterized by a different microscopic organization.

Phase transition

In order to introduce the notion of a phase transition, we must first define the notion of a *phase*. We have already introduced phase space. In this space, a *phase* is simply a region including all points corresponding to a homogeneous state of the system under analysis. As an example, let us consider again Figure 1.3 for a constant mass of water.

Phase transitions in water

At room temperature and normal pressure, water is liquid; this state may be represented by the point X_1 in the figure. If the pressure is kept constant and the temperature is increased, it is a matter of everyday experience that the water will boil, when the temperature reaches $100\,^{\circ}\mathrm{C}$ (the point X_2 in the figure), and become vapor. During this transformation, liquid water and vapor

[3] Note that here the symbol Ω does not have the same meaning as in expression (2.10).

coexist; as long as there is still liquid water the temperature of the mixture remains constant even though heat is being supplied; when all the liquid water has become vapor, the temperature of the latter begins to rise again, if heat continues to be supplied.

First-order phase transition

This type of transformation is said to be *first-order* phase transition, as we will see later on. From Figure 1.3 one may wonder what happens if the temperature and pressure are changed so as to maintain the mixed phases on the line from C to A. Point A is a "critical" point, at $T = 374\,^{\circ}\mathrm{C}$ and $P = 220$ atm. When A is reached the system transforms into a single fluid phase (Gitterman and Halpern, 2004). Point A is the end of the coexistence between the liquid and vapor phases, and a new type of transformation, namely a *second-order* phase transition occurs at that point. This type of phase transition is a continuous one.

Second-order phase transition

Similar reasoning applies to the transformation from ice to liquid water and vice versa.[4] The first classification of phase transitions in matter was proposed by Paul Ehrenfest in 1933 (see for instance Jaeger, 1998), following the discovery, the year before, of the λ-transition in liquid helium by W. H. Keesom and coworkers (Keesom and van den Ende, 1932). Ehrenfest classified phase transitions in terms of the thermodynamic quantities that present a discontinuity. The order of the transition is the same as the order of the derivative of the free energy that shows a discontinuity. New findings emerged in later decades. In order to accommodate some singularities that did not fit into Ehrenfest's scheme, this last was extended, most notably by A. Brian Pippard (1957).

Types of phase transition

More recently, T. Matolcsi classified phase transitions into three classes (Matolcsi, 1996): zeroth-order, first-order, and second-order. The order corresponds, as in Ehrenfest's classification, to the lowest-order derivative of the free energy that shows a singularity across the transition.

- **Zeroth-order phase transitions** The occurrence of this type of transition was predicted theoretically, and then experimentally verified, in the theory of superfluidity and superconductivity, where a discontinuity in the free energy itself was discovered.

- **First-order phase transitions** In this type of transition the free energy is continuous but one of its first derivatives shows a jump across the transition. Usually a large amount of energy is exchanged between the two phases but the temperature remains constant (as in the case of, say, boiling

[4] The transition between ice and water is somewhat different from that between water and vapor because the former has no critical point. Moreover, at this transition a deep restructuring of the internal organization of the matter takes place, as on the microscopic level water has a spherical symmetry, owing to its random structure, that cannot be reached smoothly from the crystalline structure of ice.

water). First-order transitions are associated with mixed-phase regimes in which the two phases coexist. In such a phase transition some other quantity may diverge. As already mentioned, in a liquid–vapor transition the specific heat diverges, since, even though heat is supplied, the temperature does not increase. Often the two phases across a first-order transition show different degrees of symmetry in their internal structure. For instance, the solid phase of water (ice) is much more symmetric than the liquid phase. The transition changes the symmetry of the internal structure. A transition from a more symmetrical phase to a less symmetrical one is said to be a *symmetry-breaking* process.

- **Second-order phase transitions** In this type of transition the free energy and its first derivatives are continuous, but there is a discontinuity in a second-order derivative of the free energy. Moreover, there is no coexistence of the phases at the transition point. Second-order transitions may be symmetry-breaking. A phenomenological theory of second-order phase transitions was presented by Landau (Landau and Lifshitz, 1980).

In modern statistical physics another type of phase transition is also considered, as follows.

- **Infinite-order phase transitions** These transitions involve no discontinuities and break no symmetries. There are examples of this kind of transition in quantum mechanics, for instance in two-dimensional electron gases.

Importance of studying phase transitions

Phase transitions are far from being limited to physical systems; on the contrary, they are an ubiquitous phenomenon, occurring in many domains including the computational, social, and biological systems. Phase transitions are strongly linked to the fundamental properties of many-body systems, and this is one reason to investigate them. Moreover the critical phenomena that occur at the transition may elucidate the very nature of the system under study. Finally, the study of phase transitions may provide answers to questions about the way in which long-range ensemble phenomena arise from local interactions.

An important observation is that a phase transition is an *asymptotic* phenomenon, i.e., it emerges when the size of the system under analysis goes to infinity. Strictly speaking, a finite system cannot have a phase transition. The reason is that, when the partition function Z is a sum of a finite number of terms (see expression (2.13)), Z itself, as well as the other thermodynamic functions, are analytic functions of the inverse temperature β and so do not have singularities at a finite temperature. Thus, the emergence of singularities in the free energy or one of its derivatives occurs when the size of the system goes to infinity, in the so-called *thermodynamic limit*.

Order parameter

In the study of phase transitions a relevant notion is that of the *order parameter*. Order parameters were defined by Landau (Landau and Lifshitz, 1980) and are a measure of the "order" of the system's state. More precisely, Landau introduced a new variable, in the expression for the free energy, which is 0 in the "disordered" phase and different from 0 in the ordered phase. The reason for the introduction of such a parameter was that in the vicinity of the phase transition (where the parameter is small) the free energy could be expanded in a Taylor series. The determination of the order parameters in a phase transition is somewhat a matter of art.

Control parameter

While the order parameter is a function that describes the changes undergoing in a phase transition, a *control parameter* is an external variable whose values determine the location of the critical point. The relevance of the control parameter is that small changes in its value cause large-scale qualitative changes in the state of the system. For instance, in the liquid–vapor transition, the density is an order parameter, whereas the control parameters are temperature and pressure.

2.4 Ising models

History of the Ising model

In order to illustrate the notions introduced so far, in this section we will describe the well-known *Ising model* (Ising, 1925), which was proposed to describe magnetization and was used in early studies on phase transitions in learning. Both Cipra (1987) and Gitterman and Halpern (2004) present a simple introduction to the mathematics of the Ising model. It may have dimension $d \geqslant 1$. The one-dimensional (1D) model was solved exactly by Ising in his doctoral thesis (Ising, 1925); it does not show a phase transition for any temperature strictly greater than 0 (Gitterman and Halpern, 2004). Probably this negative result dissuaded Ising from further study of the model. A decade later, in 1936, interest was raised again by the proof, offered by Rudolf Peierls (1936), that the two-dimensional (2D) Ising model was indeed guaranteed to have a phase transition for some temperature. Later on, Hendrick Kramers and Gregory Wannier (1941) located exactly the phase transition in the 2D model. But it was only later on that Lars Onsager (1944) provided a complete solution of the 2D Ising model in the absence of an external magnetic field. The three-dimensional (3D) model is yet unsolved, as well as the 2D model with a non-zero external field, but many partial results are available.

In the Ising model it is assumed that a number N of particles (originally, magnetic dipoles) are located on the nodes of a lattice in d dimensions, with $d = 1, 2, 3$. Each particle has a *spin* that can be in one of two states, $\sigma_i = \pm 1$

(corresponding to spin up or spin down). The particles interact with each other in pairs, according to a specified pattern. If two particles interact, the nodes corresponding to them are connected by an arc, called a *bond*. In particular there may exist short-range interactions, between pairs of nearest neighbor particles, or long-range interactions, involving particles that can be far apart. The range of the interactions determines the macroscopic behavior of the ensemble of particles. In a lattice with interactions between pairs of nearest neighbors only, each particle has $2d$ bonds.

Interaction between spins

If the lattice is finite, particles at the borders have fewer bonds than those in the middle

Let Z be the partition function of the spin system. The main quantity to be calculated, from which all the interesting behaviors of the system can be derived, is the *free energy per site*, $f = F/N$, in the thermodynamic limit:

$$f = -\lim_{N\to\infty} \frac{k_B T}{N} \ln Z. \tag{2.20}$$

The main problem in the Ising model is to find a closed-form, analytic, expression for the function f. Notice that, in principle, the limit in (2.20) may not exist.

Given two spins σ_i and σ_j, let J_{ij} be a coefficient determining the strength of their interaction. The parameter J_{ij} is called the *coupling*. The potential energy of the interaction is given by

$$E_{ij} = -J_{ij}\sigma_i\sigma_j.$$

The total energy of the system is the sum of the E_{ij} over all pairs of spins:

$$E = -\frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N} J_{ij}\sigma_i\sigma_j.$$

In the energy expression, $J_{ii} = 0$ for $1 \leqslant i \leqslant N$, and the factor $1/2$ derives from the fact that each term is counted twice.

2.4.1 One-dimensional Ising model

For the sake of simplicity, let us start with the 1D Ising model, in which N spins are arranged in a linear lattice. Let us assume that only adjacent spins interact and that the strength of the interaction, J, is constant for all pairs. Moreover, we assume that there is no external magnetic field. The total energy of the system can be computed as:

Energy of 1D Ising model

$$E(\sigma_1, \ldots, \sigma_N) = -\sum_{1=1}^{N-1} J\sigma_i\sigma_{i+1}. \tag{2.21}$$

Sometimes, in order to remove the end effects of a finite-length line of spins, an artificial bond between the last and the first spin is introduced. In this way, the

spins form a ring. The energy reaches its minimum value, $E_0 = -J(N-1)$, when all the spins point in the same direction. Using (2.13), the partition function of a system with N spins can be written as

$$Z_N = \sum_{\sigma_1=\pm1} \sum_{\sigma_2=\pm1} \cdots \sum_{\sigma_N=\pm1} \exp\left[-\frac{J}{k_B\,T}(\sigma_1\sigma_2 + \cdots + \sigma_{N-1}\sigma_N)\right]. \quad (2.22)$$

In order to calculate Z we suppose that another spin σ_{N+1}, is added at the end of the chain and compute the new partition function, Z_{N+1}, which can be related to Z_N as follows:

$$Z_{N+1} = \sum_{\sigma_1=\pm1} \sum_{\sigma_2=\pm1} \cdots \sum_{\sigma_{N+1}=\pm1} \exp\left[-\frac{J}{k_B\,T}(\sigma_1\sigma_2 + \cdots + \sigma_{N-1}\sigma_N)\right] \times \exp\left(-\frac{J}{k_B\,T}\sigma_N\sigma_{N+1}\right).$$

By rewriting the preceding expression and setting σ_{N+1} equal first to 1 and then to -1, we obtain

$$\begin{aligned} Z_{N+1} = &\sum_{\sigma_1=\pm1} \sum_{\sigma_2=\pm1} \cdots \sum_{\sigma_N=\pm1} \exp\left[-\frac{J}{k_B\,T}(\sigma_1\sigma_2 + \cdots + \sigma_{N-1}\sigma_N)\right] \times \exp\left(\frac{J}{k_B\,T}\sigma_N\right) \\ &+ \sum_{\sigma_1=\pm1} \sum_{\sigma_2=\pm1} \cdots \sum_{\sigma_N=\pm1} \exp\left[\frac{J}{k_B\,T}(\sigma_1\sigma_2 + \cdots + \sigma_{N-1}\sigma_N)\right] \times \exp\left(-\frac{J}{k_B\,T}\sigma_N\right). \end{aligned}$$

When σ_N assumes the values $+1$ and -1, the two factors $e^{J\sigma_N/k_BT}$ and $e^{-J\sigma_N/k_BT}$ interchange, so their sum can be factorized:

$$Z_{N+1} = Z_N\; 2\cosh\left(\frac{J}{k_B\,T}\right). \quad (2.23)$$

As the system cannot have just one spin (since then there would be no interaction), we will compute Z for $N = 2$:

$$\begin{aligned} Z_2 = \sum_{\sigma_1=\pm1} \sum_{\sigma_2=\pm1} \exp\left(-\frac{J}{k_BT}\sigma_1\sigma_2\right) &= 2\,\exp\left(\frac{J}{k_BT}\right) \\ &+ 2\,\exp\left(-\frac{J}{k_BT}\right) = 4\,\cosh\left(\frac{J}{k_BT}\right). \end{aligned}$$

By induction, using Z_2 as the base step and expression (2.23) as the recursive step, we finally obtain

$$Z = Z_N = 2^N \left(\cosh \frac{J}{k_B T}\right)^{N-1}. \tag{2.24}$$

Partition function of 1D Ising model

Then the free energy F can be written as follows:

$$F = -k_B T \ln Z = -k_B T \left[N \ln 2 + (N-1) \ln \cosh \left(\frac{J}{k_B T}\right)\right].$$

As $N \gg 1$, we can approximately equate N and $N-1$, obtaining finally:

$$F = -k_B T \ln Z = -k_B N T \ln \left(2 \cosh \frac{J}{k_B T}\right). \tag{2.25}$$

The free energy per site f is

Free energy per site

$$f = -k_B T \ln \left(2 \cosh \frac{J}{k_B T}\right). \tag{2.26}$$

As f is an analytic function, it cannot exhibit a phase transition for any finite value of T, except $T = 0$.

2.4.2 Two-dimensional Ising model

In order to analyse the 2D Ising model, we will consider a square lattice, at each node (site) of which a spin is located, as in Figure 2.2. We assume that each spin interacts only with its four nearest neighbors (top, bottom, left, right), and that there is no external magnetic field. Again, the strength of the interaction is constant and equal to J. Then the energy of the spin located at a point (i, j) will be

$$E(i,j) = -J\sigma_{i,j}(\sigma_{i-1,j} + \sigma_{i+1,j} + \sigma_{i,j-1} + \sigma_{i,j+1}). \tag{2.27}$$

Energy of the 2D Ising model

To obtain the total energy E we have to sum over all sites of the $N \times N$ grid:

$$\begin{aligned} E &= -J \sum_{i \neq j} E(i,j) = -J \sum_{i \neq j} \sigma_{i,j}(\sigma_{i-1,j} + \sigma_{i+1,j} + \sigma_{i,j-1} + \sigma_{i,j+1}) \\ &= -J \sum_{i=2}^{N} \sum_{j=1, j\neq i}^{N} \sigma_{i,j}\sigma_{i-1,j} - J \sum_{i=1}^{N-1} \sum_{j=1, j\neq i}^{N} \sigma_{i,j}\sigma_{i+1,j} \\ &\quad -J \sum_{i=1}^{N} \sum_{j=2, j\neq i}^{N} \sigma_{i,j}\sigma_{i,j-1} - J \sum_{i=1}^{N} \sum_{j=1, j\neq i}^{N-1} \sigma_{i,j}\sigma_{i,j+1}. \end{aligned}$$

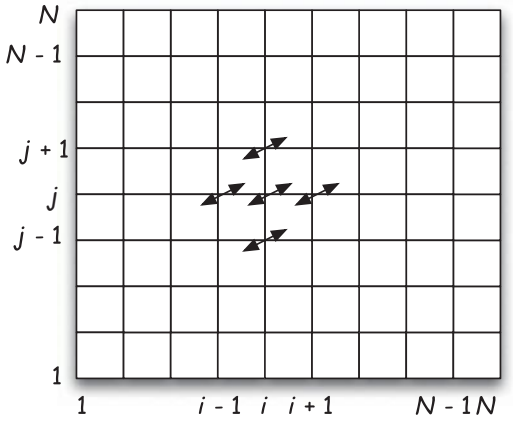


Figure 2.2 Square lattice of size N. A spin is located at each of the N^2 sites. Each spin can take two values (up, $+1$, and down, -1) and interacts only with its four nearest neighbors (except for the spins at the edges of the lattice).

As in the 1D Ising model, the four sums may take into account the border effects. Using the expression for the energy, after rather long computations the partition function and the free energy F can be obtained. At all temperatures T the stable state of the system corresponds to the minimum of F. When T is low, the energy predominates and F's minima correspond to ordered states. When T is high, the entropy dominates and the minima of F correspond to the minima of S and, hence, to disordered states. At some intermediate temperature T_{cr} the two effects balance each other; T_{cr} is the temperature of a critical point corresponding to a *phase transition*. Onsager (1944) was able to compute the value of T_{cr} exactly:

Critical value of the temperature

$$T_{cr} = \frac{2J}{k_B \ln(1+\sqrt{2})}. \tag{2.28}$$

Following an argument put forwad by Svrakic (1980) and further elaborated by Gitterman and Halpern (2004), the value given by (2.28) can be derived. To this end let us consider a cell with four sites, as represented in Figure 2.3. As each site may have up or down spin, there are 16 possible configurations of the cell, whose energy can be easily computed assuming that there are only nearest neighbor interactions and that all interactions have strength J:

$$E = -J(\sigma_1\,\sigma_2 + \sigma_2\sigma_3 + \sigma_3\sigma_4 + \sigma_4\sigma_1).$$

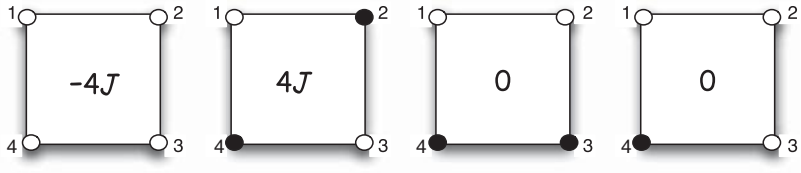


Figure 2.3 Qualitative argument allowing the value of the critical temperature T_{cr} to be computed. There are four spins, connected as described, and hence 16 possible configurations. For the configurations $\gamma_1 = (\sigma_1 = 1, \sigma_2 = 1, \sigma_3 = 1, \sigma_4 = 1)$ and $\gamma_2 = (\sigma_1 = -1, \sigma_2 = -1, \sigma_3 = -1, \sigma_4 = -1)$ the energy equals $-4J$, whereas for the configurations $\gamma_3 = (\sigma_1 = 1, \sigma_2 = -1, \sigma_3 = 1, \sigma_4 = -1)$ and $\gamma_4 = (\sigma_1 = -1, \sigma_2 = 1, \sigma_3 = -1, \sigma_4 = 1)$ the energy equals $4J$. In the other 12 configurations the energy equals zero.

Among these configurations, four correspond to "ordered" states, two with energy $4J$, and two with energy $-4J$, whereas 12 are "disordered", with energy equal to 0. For "ordered" states, it is possible to build an isotropic infinite 2D Ising lattice by repeatedly placing the basic patterns one adjacent to the other in both directions, whereas this is not possible with "disordered" states. Assuming that ordered and disordered states compete, it is possible to conjecture that the phase transition will occur when the partition function of the ordered state, namely

$$Z_{ord} = 2\ \exp\left(\frac{4J}{k_B T}\right) + 2\ \exp\left(-\frac{4J}{k_B T}\right),$$

is equal to that of the disordered state:

$$Z_{dis} = 12.$$

By defining $x = \exp\left(\frac{4J}{k_B T}\right)$, we obtain the equation

$$x + x^{-1} = 6,$$

whose solutions are $x = 3 \pm 2\sqrt{2}$. By taking $x = 3 + 2\sqrt{2}$, it must follow that

$$\exp\left(\frac{4J}{k_B T}\right) = 3 + 2\sqrt{2} \quad \Longrightarrow \quad \frac{4J}{k_B T} = \ln(3 + 2\sqrt{2}).$$

Finally, the critical temperature is

$$T_{cr} = \frac{4J}{k_B \ln(3 + 2\sqrt{2})}. \tag{2.29}$$

Expression (2.29) is equal to the value (2.28) computed by Onsager: equating the two expressions we obtain

$$\frac{2J}{k_B \ln(1+\sqrt{2})} = \frac{4J}{k_B \ln(3+2\sqrt{2})} \quad \Longrightarrow \quad \ln(1+\sqrt{2}) = \frac{1}{2}\ln(3+2\sqrt{2})$$

$$\Longrightarrow 1+\sqrt{2} = \sqrt{(3+\sqrt{8})} = \sqrt{\frac{3+\sqrt{9-1}}{2}} + \sqrt{\frac{3-\sqrt{9-1}}{2}} = 1+\sqrt{2}.$$

Unfortunately this argument does not work for the 3D Ising model. Gitterman and Halpern, however, introduced a modified criterion to find T_{cr} in the 3D model using an argument involving a single cubic cell. In this way they found an estimate of the critical temperature, $T_{cr} = 4.277\ J/k_B$, that is close to the numerical result $T_{cr} = 4.511\ J/k_B$ found by Binder and Luijten (2001) using Monte Carlo simulations.

2.5 Mean field theory

Computing the behavior of a system of many interacting particles (an N-body system) is a problem that is difficult to solve except in very simple cases. The pattern of interaction makes the evaluation of the partition function a hard combinatorial task, owing to the need to sum over all the states of the system. A possible way out of the difficulty is to use a mean field approach. The basic idea is to replace the N-body system, with all its interactions, by a one-body system immersed in an appropriate external field. In this way an approximate, but feasible, solution can be obtained. The external field replaces the interaction of all the other particles, for an arbitrary particle. In this way some insight into the behavior of the system can be obtained at a relatively low cost.

Mean field theory

The central idea of *mean field theory* (MFT) is to neglect the fluctuations of microscopic quantities around their average. For example, let us consider a set

No external field

of N spins σ_i, in the absence of an external field, and let $\mathcal{B}$ be the set of bonds between the spins. Moreover, let $|\mathcal{B}| = N_B$. Each spin value can be written as the sum of its average m (over all the spins) and a fluctuation $\delta\sigma_i = \sigma_i - m$ around the average:

$$\sigma_i = m + \delta\sigma_i,$$
$$m = \frac{1}{N}\sum_{i=1}^{N} \sigma_i.$$

In MFT it is assumed that the second-order term in the fluctuation is negligible in the computation of the Hamiltonian: Hamiltonian

$$\begin{aligned} H &= -J \sum_{(ij)\in\mathcal{B}} \sigma_i\sigma_j = -J \sum_{(ij)\in\mathcal{B}} (m+\delta\sigma_i)(m+\delta\sigma_j) \\ &= -J \sum_{(ij)\in\mathcal{B}} [m^2 + m(\delta\sigma_i + \delta\sigma_j) + \delta\sigma_i\delta\sigma_j] \\ &\simeq -Jm^2 N_B - Jm \sum_{(ij)\in\mathcal{B}} (\delta\sigma_i + \delta\sigma_j). \end{aligned}$$

In order to compute the last sum we notice that each bond occurs just once in the sum and that each deviation occurs as many times as the number of bonds connected to each site. If z is the number of bonds per site then The number of bonds per site is z.

$$H = -Jm^2 N_B - Jmz \sum_{i=1}^{N} \delta\sigma_i = Jm^2 N_B - Jmz \sum_{i=1}^{N} \sigma_i. \tag{2.30}$$

In equation (2.30) the parameter z has been assumed to be independent of the site; then $N_B = zN/2$. The interactions between the spins are embedded in the average m. The MFT can be considered as a zeroth-order expansion of the Hamiltonian in terms of the fluctuations. Starting from it, the analysis of first- and second-order approximations can be investigated.

The approximate Hamiltonian allows other physical quantities to be computed more easily. For instance, the partition function can be expressed as Approximate partition function

$$\begin{aligned} Z &= \sum_{i=1}^{N} \exp\left[\beta\left(Jm^2 N_B - Jmz \sum_{i=1}^{N} \sigma_i\right)\right] \\ &= \exp\left(-\beta N_B J m^2\right) \left[2\cosh(\beta Jmz)\right]^N. \end{aligned} \tag{2.31}$$

An important point in MFT is to estimate whether a mean field approach is a good approximation for any particular problem. From the statistical perspective, a system with many interactions is a good candidate for an appropriate approximation.

2.6 Quenched disorder and self-averaging

In the previous sections we considered systems whose parameters were given, as, for instance, the coupling J of the spin interaction in equation (2.21). We will now consider the case in which the system's parameters, even though constant in time, are random variables whose values follow a known probability distribution. In this case the system is said to exibit *quenched disorder*. The term "quenched", here meaning "frozen", refers to the invariance of the parameters with respect to time. A "quench" refers to a rapid cooling. In metallurgy, such a process is commonly used to harden steel.

In the presence of quenched disorder the system's parameters are stochastic variables.

Clearly, the introduction of a further probabilistic aspect in a system makes the mathematical analysis of its behavior and properties much more complex than in the purely deterministic case. In the presence of quenched disorder the structure of the system becomes random, and the weights in the partition function are themselves random variables (Martin *et al.*, 2001). Now the partition function and the other physical quantities depend on two types of randomness, a thermal randomness, whose distribution could be, for instance, the Gibbs distribution, and a structural randomness, whose distribution we will denote by ρ. When computing thermal average values, we keep the quenched variables fixed and then average the results with respect to ρ. In order to distinguish these two averages, we will use *angle brackets* to indicate a thermal average and an *overbar* to denote an average over the quenched random variables.

EXAMPLE

Following Martin *et al.* (2001), we will consider, as an example, a system of two spins σ_1 and σ_2, with energy

$$E(J) = -J\sigma_1\sigma_2.$$

The system can be in one of four states $\{s_1, s_2, s_3, s_4\}$, where

$$s_1 = \{\sigma_1 = 1, \sigma_2 = 1\}, \qquad s_2 = \{\sigma_1 = 1, \sigma_2 = -1\},$$
$$s_3 = \{\sigma_1 = -1, \sigma_2 = 1\}, \qquad s_4 = \{\sigma_1 = -1, \sigma_2 = -1\}.$$

Correspondingly, the energy assumes the values

$$E(J, s_1) = -J, \quad E(J, s_2) = J, \quad E(J, s_3) = J, \quad E(J, s_4) = -J.$$

Then, according to formula (2.13), the partition function can be computed as follows:

$$Z(J) = 2\ \exp\left(\frac{J}{k_B T}\right) + 2\ \exp\left(-\frac{J}{k_B T}\right) = 4\ \cosh\left(\frac{J}{k_B T}\right).$$

Setting $\beta = 1/k_B T$ as usual, we deduce from (2.16) that

$$\langle E(J)\rangle = -\frac{\partial}{\partial\beta}\ \ln Z = -J\ \tanh\ \beta J.$$

Let us assume that the coupling J is a random variable that can take on two values $\{J_0, J_1\}$ according to a probability distribution $\rho(J)$ such that $\rho(J_0) = \rho_0$ and $\rho(J_1) = 1 - \rho_0$. By averaging the thermal average energy with respect to J, we obtain

$$\overline{\langle E\rangle} = -\rho_0 J_0 \tanh\beta J_0 - (1 - \rho_0) J_1 \tanh\beta J_1.$$

If we assume that $J_0 = -1$, $J_1 = 1$, and $\rho_0 = 1/2$, we finally obtain

$$\overline{\langle E\rangle} = -\tanh\beta.$$

In order to analyze quenched disorder it is useful to introduce the notion of *frustration*. Considering again a system of spins, let us suppose that the coupling may assume both positive and negative values. In this case the minimum of the energy is not reached when all spins point in the same direction, and "conflicts" between different configurations arise; the system is said to be "frustrated". Frustration

EXAMPLE

Let us consider a set of N spins, located at the vertices of a complete graph (a graph in which every pair of vertices is connected by a unique edge), and let $\{J_{ij}\}$ be the set of couplings, which can assume both positive and negative values. The J_{ij} are independent random variables following a known probability distribution. This model is the famous *Sherrington–Kirkpatrick* (SK) model (Sherrington and Kirkpatrick, 1975), which is a stochastic version of Ising's model. The SK model has energy

$$E_{SK} = -\frac{1}{\sqrt{N}} \sum_{i=1}^{N-1} \sum_{j=i+1}^{N} J_{ij}\sigma_i\sigma_j .$$

Owing to the conflicts in E_{SK} arising from the presence of both positive and negative terms, to find the ground state energy for arbitrary J_{ij} is an **NP**-complete problem (Cipra, 2000).

Because of the mathematical complexity of handling quenched disorder, approximate methods such as the *replica method* (Mézard and Parisi, 1991) and the *cavity method* (Mézard and Parisi, 1985) have been developed. These methods provide results that are very often in agreement with experiment, but they have not been proved exact in general; they will be illustrated briefly in the following subsections.

We have seen that in the SK model the Ising model was extended in two ways: on the one hand by introducing quenched disorder and on the other by changing the geometry of the underlying interaction topology, so that, rather than being on a regular lattice, the spins are located at the vertices of a complete graph. It is not surprising, then, that other topologies have been investigated. For instance, instead of a complete graph one may think of using a graph which is random in respect of the number of edges (Hartmann and Weigt, 2005; Barré *et al.*, 2009). The topology of the structure is important because it may affect the behavior of the system, in particular the presence and location of phase transitions.

Along the same line, other topologies have been investigated. For instance, Jeong *et al.* (2003) studied the Ising model on a small-world network (Watts and Small-world network.

Strogatz, 1998). They assumed an interaction of the type

$$J_{ij} = J_{ji} = \begin{cases} J r_{ij}^{-\alpha} & \text{if } i \text{ and } j \text{ are connected,} \\ 0 & \text{otherwise,} \end{cases}$$

where r is the Euclidean distance between two nodes. The authors found that, for an interaction $\propto r^{-\alpha}$, there is no phase transition for any non-zero positive value of α.

More recently, Brunson and Boettcher (2009) studied the Ising model on a new network, with small-world properties, that can be studied exactly using the renormalization group. The network is non-planar and has a recursive design that combines a one-dimensional backbone with a hierarchy of long-range bonds. By varying the relative strength of nearest-neighbor and long-range bonds, the authors were able to defined a one-parameter family of models that exhibits a rich variety of critical phenomena quite distinct from those on lattice models.

Scale-free network

Finally, systems of spins on scale-free networks (Barabási and Bonabeau, 2003) have also been investigated. For instance, Herrero (2009) studied uncorrelated networks using Monte Carlo simulation (Durrett, 2007).

2.6.1 Self-averaging quantities

Even though physical quantities depending on quenched parameters are difficult to compute, their behavior in the *thermodynamic limit* ($N \to \infty$) may show the important *self-averaging* property. In other words, as the size N of the system increases, such a quantity shows a decreasing amplitude of fluctuation around its mean, so that its values become highly concentrated around the mean. More precisely, in the thermodynamic limit the value of a self-averaging quantity is equal to its mean with probability 1. Typical examples of self-averaging quantities are the free energy, the entropy, and the magnetization; the partition function itself is generally not self-averaging.

Self-averaging quantity

The importance of self-averaging is that in the case of large systems it is not necessary to compute the whole distribution of values for self-averaging quantities; the mean is sufficient. In fact, self-averaging quantities do not depend on the particular type of quenched disorder exhibited by the system, but only on the statistical properties of their distribution. In formal terms we can write, for the free energy density,

Free energy density

$$f = -\frac{1}{\beta} \lim_{N\to\infty} \frac{1}{N} \overline{\ln Z(J)}. \tag{2.32}$$

Expression (2.32) indicates that it is possible to first calculate the disorder average for systems of finite size and then take the thermodynamic limit. The self-averaging property is restricted to *intensive* quantities, i.e., to quantities whose

value does not depend on the amount of substance for which they are computed. For instance, the density of an object is an intensive quantity whereas its mass is an *extensive* quantity since it is proportional to the amount of substance.

2.7 Replica method

The *replica approach* is a heuristic procedure used to simplify the hard computation involved in obtaining the value of the partition function and other thermodynamical quantities in the presence of quenched disorder. The method is reported to have been first proposed by M. Kac in a seminar in the 1950s (Kac and Ward, 1952), but its formalization was worked out by Mézard and Parisi (Mézard *et al.*, 1987; Mézard and Parisi, 1991). Kac's proposal

For exemplification purposes, let us consider a system composed of N spins and let a set J of couplings represent its quenched disorder. Let $Z(J)$ be the partition function. The replica method starts from the equality

$$\overline{\ln Z(J)} = \lim_{r \to 0} \frac{\overline{Z(J)^r} - 1}{r}, \tag{2.33}$$

which holds in a composite system formed by r identical and independent replicas of the system under consideration; in (2.33) an overbar denotes an average over the replicas. Expression (2.33) derives from the expansion

$$Z(J)^r = 1 + r \ln Z(J) + \mathcal{O}(r^2), \tag{2.34}$$

valid for any set of couplings J and small r. Remembering that the logarithm of the partition function is linked to the free energy by $F(J) = -k_B T \ln Z(J)$, we obtain $\ln Z(J) = -F(J)/k_B T$. Inserting this expression into (2.34) we obtain:

$$Z(J)^r = 1 - r \, \frac{F(J)}{k_B T} + \mathcal{O}(r^2).$$

By taking the average over J, ignoring the term $\mathcal{O}(r^2)$, letting r go to 0, and using (2.33) we finally obtain

$$\overline{F(J)} = -k_B T \lim_{r \to 0} \left(\frac{\overline{Z(J)^r} - 1}{r} \right). \tag{2.35}$$

The intuition behind the preceding derivation can be explained as follows. Let us suppose that we have not one, but r replicas of the same system, all with the same quenched disorder. We have now to compute $\overline{Z(J)^r}$. Then, we apply the *replica trick*, i.e., we can compute the partition function of the composite system Replica trick

as the product of the partition functions of the components:

$$\begin{aligned} Z(J)^r &= Z(J)Z(J)Z(J)\cdots Z(J) \\ &= \left(\sum_{\{\sigma_i^1\}} \exp\left[-\beta H\left(\{\sigma_i^1\}\right)\right]\right)\cdots\left(\sum_{\{\sigma_i^r\}} \exp\left[-\beta H\left(\{\sigma_i^r\}\right)\right]\right) \\ &= \sum_{\{\sigma_i^a\}a=1,\ldots,r} \exp\left(-\beta\sum_{a=1}^{r} H(\{\sigma_i^a\})\right) \\ &= \sum_{\{\sigma_i^a\}} \exp\left(\beta\sum_{i<j} J_{ij}\sum_{a=1}^{r}\sigma_i^a\sigma_j^a\right). \end{aligned}$$

If we restrict ourselves to integer r, the rth moment of the partition function Z can be rewritten as

$$\overline{Z(J)^r} = \overline{\left(\sum_{\gamma\in\Gamma} e^{-\frac{1}{T}E(\gamma,J)}\right)^r} = \sum_{\gamma^{(1)},\ldots,\gamma^{(r)}\in\Gamma^r} \overline{e^{-\frac{1}{T}\sum_{a=1}^{r}E(\gamma^{(a)},J)}}, \qquad (2.36)$$

where Γ is the configuration set. The idea is that the randomness of the couplings disappears once the averaging process has been carried out. Considering the r replicas as a composite system, this last contains N vectors $\vec{\sigma_j}$ of spins, one for each original site. Thus each vector has r components, each corresponding to a replica of the spin at that site:

$$\vec{\sigma_j} = \left\{\sigma_j^{(1)},\ldots,\sigma_j^{(r)}\right\} \qquad (1\leqslant j\leqslant N).$$

We can now compute the partition function for the composite system of N vectorial spins using the non-random energy function

$$E(\vec{\sigma_j}) = -k_B T \ln\exp\left[\overline{-\frac{1}{T}\sum_{a=1}^{r} E\left(\gamma_j^{(a)},J\right)}\right], \qquad (2.37)$$

where γ_j is the configuration that corresponds to the spin vector $\vec{\sigma_j}$.

Analytic continuation

The new partition function can be estimated analytically, in some cases, by means of the *saddle-point* method. The idea is to compute the right-hand side of (2.35) and take the limit for $r\to 0$ by using *analytic continuation*. The problem is that the analytic continuation is not unique and a heuristic hypothesis, an *ansatz*, on the mathematical structure is needed. To solve the problem, Parisi proposed a complete and internally consistent *ansatz* scheme, called *replica symmetry* (Mézard and Parisi, 1985). This *ansatz*, although very useful in particular applications, has not yet been proven to be correct in general.

Replica symmetry *ansatz*

Recently, Wästlund proved the soundness of the replica symmetry *ansatz* for the minimum-matching and traveling-salesman problems in the pseudo-dimension-d mean field model for $d \geqslant 1$ (Wästlund, 2009). Also, Krzcakala used this *ansatz* in the problem of graph coloring with q colors and argued that in this case the *ansatz* gives exact threshold values between the colorable and uncolorable phases (Krzcakala, 2005).

2.8 Cavity method

The *cavity* method was proposed by Mézard, Parisi, and Virasoro in 1985 as a tool for solving mean field models that was particularly well suited to handling disordered systems (Mézard and Parisi, 1985). Even though it was designed with the Sherrington–Kirkpatrick model of spin glasses (Sherrington and Kirkpatrick, 1975) in mind, the method has proved to be applicable to a wide range of problems. In particular, the method has been used in computer science problems, notably k-SAT (see for instance Mertens *et al.*, 2006) and graph coloring (see for instance Krzcakala, 2005; Mulet *et al.*, 2002). Even though the cavity approach is in principle equivalent to the replica method, the former has a more intuitive interpretation and is easier to apply, so that it is often preferred.

Bethe lattice

In order to explain how the method works, let us consider a particular system of Ising spins, defined on a *Bethe lattice* (Bethe, 1935). A Bethe lattice is a random graph with N vertices and fixed connectivity equal to $k+1$, i.e., it is a graph in which there are $k+1$ incoming edges at each site. Spins are located at the vertices of the graph and interact with neighboring spins only. In such a spin system disorder is due to the presence of loops (see Figure 2.4) and thus occurs only on large scales. The local structure around each site is tree-like. As usual, the Hamiltonian is defined as

$$H(J) = -\sum_{i<j} J_{i,j}\sigma_i\sigma_j,$$

where the J_{ij} are independent identically distributed random variables with probability distribution $\rho(J)$. Our goal is to find the value of the energy density u of the ground state in the thermodynamic limit, $N \to \infty$. More precisely, we want to compute the ground state energy, averaged both over the distribution of random graphs and the values of the couplings. This energy is denoted by $E_0^{(N)}$. We want to compute:

Ground state energy density

$$u = \lim_{N\to\infty} \frac{E_0^{(N)}}{N}.$$

The cavity method works iteratively on a spin model with N spins, defined on a slightly different graph $\mathcal{G}(N, q)$, which is derived from the Bethe lattice and

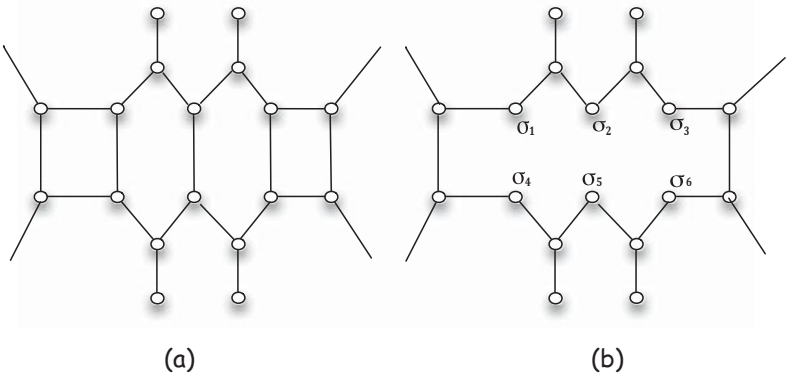


Figure 2.4 (a) Example of a part of a Bethe lattice with $k = 2$ and N vertices. Each vertex has exactly $k + 1 = 3$ bonds. (b) The cavity graph $\mathcal{G}(N, 6)$ obtained from the Bethe lattice in (a). Here $q = 6$ and the removed bonds are such that six spins have now exactly two bonds. Notice that the sites with a bond removed are randomly selected. In the figure they have been drawn as adjacent for the sake of simplicity.

Cavity graph

called a *cavity graph*. A cavity graph is obtained by removing some bonds from the original graph, in such a way that q randomly chosen "cavity" spins have only k neighbors while the other $N - q$ spins all have $k + 1$ neighbors, as illustrated in Figure 2.4. Each removed bond generates a "cavity" in the original graph. The cavity spins have fixed values $\sigma_1, \ldots, \sigma_q$. The global ground state energy of the corresponding spin model depends on the values of these cavity spins. The basic operations that one can perform on cavity graphs (illustrated in Figure 2.5) are the following.

- **Iteration** This operation consists in adding a new spin σ_0 of fixed value to the cavity, and connecting it to k of the cavity spins, say $\sigma_1, \ldots, \sigma_k$. Thus the graph $\mathcal{G}(N, q)$ becomes $\mathcal{G}(N + 1, q - k + 1)$. The increments in the number of vertices and in q are as follows:

$$\delta N = 1, \qquad \delta q = -k + 1.$$

- **Link addition** This operation consists in adding a new link between two randomly chosen cavity spins σ_1, σ_2. The graph $\mathcal{G}(N, q)$ becomes $\mathcal{G}(N, q - 2)$. The increments in the number of vertices and in q are as follows:

$$\delta N = 0, \qquad \delta q = -2.$$

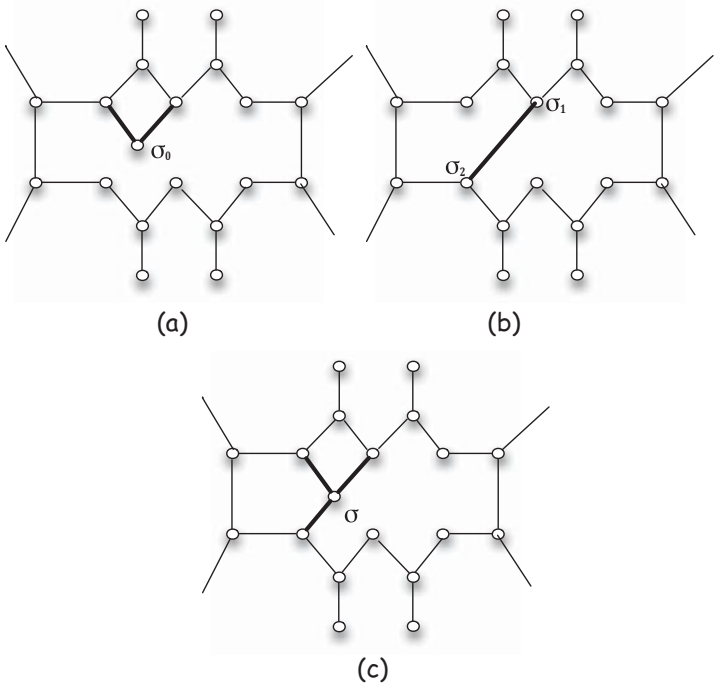


Figure 2.5 (a) Iteration. A new spin σ_0 is added and connected to $k = 2$ randomly chosen cavity spins. (b) Link addition. A new bond is added between two randomly chosen cavity spins σ_1 and σ_2. (c) Site addition. A new spin σ_0 is added and connected to $k + 1 = 3$ randomly chosen cavity spins.

- **Site addition** This operation consists in adding a new spin σ_0 to the cavity and connecting it to $k + 1$ of the cavity spins, say $\sigma_1, \ldots, \sigma_{k+1}$. The graph $\mathcal{G}(N, q)$ now becomes $\mathcal{G}(N + 1, q - k - 1)$. The increments in the number of vertices and in q are as follows:

$$\delta N = 1, \qquad \delta q = -k - 1.$$

Now, if in a cavity graph $\mathcal{G}(N, 2(k + 1))$ a number $k + 1$ of link additions are performed then a $\mathcal{G}(N, 0)$ graph is obtained; $\mathcal{G}(N, 0)$ is nothing other than the original spin glass problem with N spins. As an alternative, starting from the same cavity graph $\mathcal{G}(N, 2(k + 1))$ and performing two site additions, a $\mathcal{G}(N + 2, 0)$ graph is obtained, which is the original spin glass problem with $N + 2$ spins. We can express the difference $E_0^{(N+2)} - E_0^{(N)}$ in the ground state energies in going from N to $N + 2$ in terms of the difference in energy due to a site

addition ΔE_s and the difference in energy due to a link addition ΔE_ℓ:

$$E_0^{(N+2)} - E_0^{(N)} = 2\Delta E_s - (k+1)\,\Delta E_\ell. \tag{2.38}$$

As the energy is an extensive quantity its asymptotic behavior is linear in N, so that

$$u = \lim_{N\to\infty} \frac{E_0^{(N)}}{N} = \frac{E_0^{(N+2)} - E_0^{(N)}}{2} = \Delta E_s - \frac{k+1}{2}\,\Delta E_\ell. \tag{2.39}$$

If $k+1$ is even, the fraction $(k+1)/2$ is an integer. Then, formula (2.39) tells us that in order to increase N by 1 it is necessary to remove $(k+1)/2$ links and then add one site. Notice that the energy required to remove a link is the opposite of the energy required to add a link.

2.9 Comments

Even though the content of this chapter looks quite far from the main theme of this book, the methods outlined in it have strong connections with computational issues, especially with the solving of hard combinatorial problems in general and with machine learning in particular. Over a long period, connections between these fields have been occasionally established and have brought innovative solutions and deep insights into the basic computational aspects of problem solving. This reason, and also the suggestion coming from statistical physics approaches of new, very effective, algorithms, has encouraged computer scientists to look more and more frequently for analogies and parallels with physical systems. In subsequent chapters, several instances of such connections, not only with computer science but also with other domains such as the social and cognitive sciences, will be presented.

From the practical point of view, for a computer scientist wanting to use the approaches mentioned in this chapter it is not necessary to become deeply acquainted with statistical physics; some basic notions, complemented by the mathematical tools sketched in this chapter, will be sufficient to apply these approaches fruitfully. Useful introductory books include Percus *et al.* (2006) and Hartmann and Weigt (2005).

3

The satisfiability problem

Contents

3.1 General framework

Surprising as it may be, the emergence of phase transitions is not limited to physical systems: it seems to be a rather ubiquitous phenomenon, existing in biology, genetics, neural networks, complex systems, and also in combinatorial problems. For the last, a precise parallel can be established with physical systems composed of very large numbers of particles. In a combinatorial problem the phase transition concerns the behavior of some *order parameter* (usually the expectation value of a microscopic quantity) characterizing an aspect of the system (often the probability of existence of a solution). Moreover, in correspondence to the phase transition (at a critical value of the control parameter), a large increase in the computational complexity of the algorithm used to find a solution is usually observed.

The key concept that allows ideas and methods to be transferred from statistical physics to combinatorial optimization and decision problems is *randomness*. If problem instances are taken in isolation, the application of these methods does

not make sense. They are only meaningful if applied to a set, i.e., an *ensemble*, of problem instances whose probability of occurrence is governed by a well defined law. Then the results obtained can be considered valid for any randomly extracted element from the ensemble. It is thus necessary to specify exactly how the elements of the ensemble are constructed.

Random graphs

The study of random graphs was pioneered by Erdös and Rényi (1959). Studying the clustering of vertices, they found that, by varying the parameters that controlled the connectivity of a random graph, in the neighborhood of a particular value there was a sudden transition from a state in which the graph had many disconnected components to a state in which it had collapsed into a single connected component involving most of its nodes. They called this behavior a *zero–one* law. This law can be said to describe in modern terms a phase transition in connectivity. Since then, the subject has raised considerable interest in the artificial intelligence (AI) community, and many works have been published on phase transitions in computer science, notably in graph partitioning (Fu and Anderson, 1986), the characterization of hard instances in CNF (Purdom and Brown, 1987), the discovery of optimal paths in trees (Karp and Pearl, 1983), and in graph coloring (Turner, 1988). A great impulse to the investigation of phase transitions in combinatorial problems was given by Cheeseman *et al.* (1991), who noticed that in the proximity of the phase transitions of several **NP** problem classes were located the most difficult instances of the class. Knowing where the most difficult instances of a problem are located is crucial in the testing of algorithms. For instance, Goldberg (1979) claimed that the majority of satisfiability problems could be solved by local search algorithms in at most $\mathcal{O}(n^2)$ time, where n is the number of variables. However, Franco and Paull (1983) showed that the probability distribution used by Goldberg to sample problem instances favored easy ones to such an extent that random assignments of truth values to the variables were able to solve the problem with probability 1.

As in physics, in computational problems also phase transitions only actually occur in systems with infinite size, and corrections to the infinite model are necessary to predict the properties of finite-size systems.

Search

A domain in which the analogy with statistical physics and the emergence of phase transitions have received most attention is *search*, probably because of its ubiquity in AI problems. A clear introductory approach to the subject is provided by Hogg *et al.* (1996). The problems most studied in both computer science and statistical physics are SAT (satisfiability) problems and CSPs (constraint satisfaction problems).

SAT and CSP

SAT problems and CSPs

Such problems are **NP**-complete. As mentioned in Chapter 1, the notion of a computational complexity based on worst-case analysis may not be very useful in practice. In fact, many instances of **NP**-complete problems are easy to solve (Cheeseman *et al.*, 1991) and thus provide evidence that reasoning based

on the use of a "typical" complexity in harder cases is correct. The phase transition framework offers a way to study problems in the "typical" complexity case which is more representative of real-world applications and also contributes to the design of efficient algorithms.

As mentioned earlier, the notion of a *problem ensemble* grounds the link between statistical physical systems and combinatorial problems. All the properties derived from the phase transition framework are valid for sets of *random* problems, whose generative model is precisely specified. Thus, an important aspect to be discussed, within the framework, is what model to use and how the chosen model can cover problems encountered in the real world. Problem ensemble

Before entering into a description of the phase transition in SAT, some preliminary notions need to be introduced.

3.2 Random graphs

Graphs are a natural way of representing the internal structure of complex systems, and they will be used extensively in the rest of the book. As previously discussed, we are not interested in single graphs but in *graph ensembles*, whose elements have an associated known probability distribution.

Definition 3.1 $\{$***Graph***$\}$ A *graph* is a pair $\boldsymbol{G} = (\mathbf{V}, \mathbf{E})$, where $\mathbf{V} = \{v_1, \ldots, v_n\}$ is a set of n vertices (or nodes) and $\mathbf{E} = \{(v_{i_1}, v_{j_1}), \ldots, (v_{i_r}, v_{j_r})\}$ is a set of r edges connecting pairs of vertices. Graph

A graph is *directed* if the edges have an orientation and is *undirected* otherwise. A graph is *weighted* if there is a function $w : \mathbf{E} \rightarrow \mathbb{R}$ that associates a real number with every edge in the graph and is *unweighted* otherwise.

Definition 3.2 $\{$***Path***$\}$ A *path* in a graph is a sequence of nodes $(v_{i_1}, \ldots, v_{i_k}, v_{i_{k+1}}, \ldots, v_{i_s})$ such that each pair $(v_{i_k}, v_{i_{k+1}})$ $(1 \leqslant k \leqslant s-1)$ belongs to the set of edges $\mathbf{E}$. The nodes v_{i_1} and v_{i_s} are said to be *connected*. If the graph is directed then v_{i_s} is said to be reachable from v_{i_1}. Path

Definition 3.3 $\{$***Connected component***$\}$ Given an undirected graph $\boldsymbol{G}$, a *connected component* in $\boldsymbol{G}$ is a maximal subgraph $\boldsymbol{G'}$ such that any two vertices in $\boldsymbol{G'}$ are connected. The subgraph $\boldsymbol{G'}$ is maximal in the sense that it is not possible to add any vertex while preserving its connectivity. Connected component

Definition 3.4 $\{$***Strongly connected component***$\}$ Given a directed graph $\boldsymbol{G}$, a *strongly connected component* in $\boldsymbol{G}$ is a maximal subgraph $\boldsymbol{G'}$ such that every vertex in $\boldsymbol{G'}$ is reachable from any other vertex. This condition requires that, given any two vertices v_{i_h} and v_{i_k} in $\boldsymbol{G'}$, v_{i_h} is reachable from v_{i_k} and vice Strongly connected component

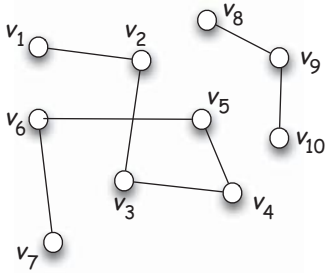


Figure 3.1 Example of an undirected graph $\boldsymbol{G} = (\mathbf{V}, \mathbf{E})$, where the set of vertices is $\mathbf{V} = \{v_1, v_2, v_3, v_4, v_5, v_6, v_7, v_8, v_9, v_{10}\}$ and the set of edges is $\mathbf{E} = \{(v_1, v_2), (v_2, v_3), (v_3, v_4), (v_4, v_5), (v_5, v_6), (v_6, v_7), (v_8, v_9), (v_9, v_{10})\}$. The sequence of vertices $(v_1, v_2, v_3, v_4, v_5, v_6)$ is a path in $\boldsymbol{G}$, and the subgraph $\boldsymbol{G}' = (\mathbf{V}', \mathbf{E}')$, where $\mathbf{V}' = \{v_8, v_9, v_{10}\}$ and $\mathbf{E}' = \{(v_8, v_9), (v_9, v_{10})\}$, is a connected component.

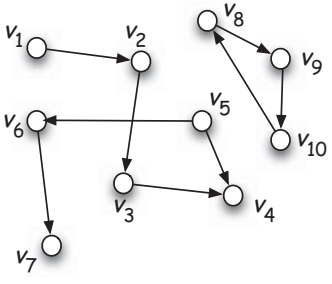


Figure 3.2 Example of a directed graph $\boldsymbol{G} = (\mathbf{V}, \mathbf{E})$, where the set of vertices is $\mathbf{V} = \{v_1, v_2, v_3, v_4, v_5, v_6, v_7, v_8, v_9, v_{10}\}$ and the set of edges is $\mathbf{E} = \{(v_1, v_2), (v_2, v_3), (v_3, v_4), (v_5, v_4), (v_5, v_6), (v_6, v_7), (v_8, v_9), (v_9, v_{10}), (v_{10}, v_8)\}$. The sequence of vertices (v_1, v_2, v_3, v_4) is a path in $\boldsymbol{G}$, and the subgraph $\boldsymbol{G}' = (\mathbf{V}', \mathbf{E}')$, where $\mathbf{V}' = \{v_8, v_9, v_{10}\}$ and $\mathbf{E}' = \{(v_8, v_9), (v_9, v_{10}), (v_{10}, v_8)\}$, is a strongly connected component.

versa. Moreover, $\boldsymbol{G}'$ is maximal in the sense that it is not possible to add any vertex while preserving its connectivity.

In Figures 3.1 and 3.2 the above definitions are illustrated for an undirected and a directed graph, respectively. Let us introduce now the notion of a random graph.

Definition 3.5 {***Random graph***} A graph $\boldsymbol{G} = (\mathbf{V}, \mathbf{E})$ is *random* if some element of its structure is built up stochastically, according to a known probability distribution. Random graph

The first model of random graphs was proposed independently by Solomonoff and Rapoport (1951) and by Erdös and Rényi (1959), and it can be formulated in two slightly different ways.

Definition 3.6 {***Model* $\mathbf{ER_1}$**}** Let $\mathcal{G}_{n,p}$ be an ensemble of random graphs $\boldsymbol{G}$, built up as follows. Given n vertices and a probability value $p \in (0, 1)$, each pair of vertices in $\boldsymbol{G}$ is connected with probability p. Model $\mathbf{ER_1}$

Definition 3.7 {***Model* $\mathbf{ER_2}$**}** Let $\mathcal{G}_{n,r}$ be an ensemble of random graphs $\boldsymbol{G}$, built up as follows. Given n vertices and an integer number r of edges, select randomly, without replacement, r pairs of vertices and connect them. Model $\mathbf{ER_2}$

In the following, let $M = n(n-1)/2$ be the total number of possible edges in an undirected graph with n vertices. In model $\mathbf{ER_1}$ the ensemble $\mathcal{G}_{n,p}$ has cardinality (number of elements)

$$|\mathcal{G}_{n,p}| = \sum_{r=0}^{M} \binom{M}{r},$$

because a graph with r edges can be chosen in $\binom{M}{r}$ different ways from the M possible graphs with r edges. A graph $\boldsymbol{G}$ with r edges occurs in $\mathcal{G}_{n,p}$ with probability

$$\mathtt{P}(\boldsymbol{G}\ has\ r\ edges) = p^r(1-p)^{M-r}.$$

In model $\mathbf{ER_2}$ the ensemble $\mathcal{G}_{n,r}$ has cardinality $|\mathcal{G}_{n,r}| = \binom{M}{r}$. All graphs in it have exactly r edges, and each occurs with uniform probability

$$\mathtt{P}(\boldsymbol{G}) = \frac{1}{\binom{M}{r}}.$$

It should be clear that $\mathcal{G}_{n,r} \subset \mathcal{G}_{n,p}$.

Definition 3.8 {***Degree***} In a graph $\boldsymbol{G}$ the *degree of a vertex* is the number of nodes connected to it. The *degree of the graph* is the maximum degree of the nodes. Degree

In a graph $\boldsymbol{G} \in \mathcal{G}_{n,p}$ the probability that a random node has degree d is given by

$$p_d = \binom{n-1}{d} p^d (1-p)^{n-d-1} \simeq \frac{z^d e^{-z}}{d!}, \tag{3.1}$$

where $z = p(n-1)$ is the average degree of the nodes. The rightmost expression is the *Poisson* approximation of the distribution p_d when $n \to \infty$ with z constant.

This approximation is valid when $n \to \infty$ while pn is kept constant. Because of the distribution (3.1) the graphs in $\mathcal{G}_{n,p}$ are called *Poisson random graphs*. Analogous properties hold for the ensemble $\mathcal{G}_{n,r}$.

Phase transition in connectivity

The ensemble $\mathcal{G}_{n,p}$ shows a second-order phase transition in the connectivity, in correspondence with the control parameter z. More precisely, when $z < 1$, a random element $\boldsymbol{G}$ of the ensemble is formed by many small disconnected components, whose size is $\mathcal{O}(\ln n)$; when $z = 1$ the mean size of the largest component becomes $\mathcal{O}(n^{2/3})$; for $z > 1$ a *giant component*, whose size is $\mathcal{O}(n)$, appears in the graph. Thus the ensemble shows, at the phase transition, a "double jump".

An ensemble of graphs that has close connections with statistical physics was proposed by Strauss (1986) and is called the ensemble of *exponential random graphs*.

Exponential random graph

Definition 3.9 {***Exponential random graph***} Given a number n of nodes, let $\epsilon_1, \epsilon_2, \ldots, \epsilon_s$ be a set of properties measured on a single graph (such as the number of edges, number of triangles, ...). Moreover, let $\beta_1, \beta_2, \ldots, \beta_s$ be a set of parameters (the *inverse temperatures*). Then the ensemble of *exponential random graphs* is a collection $\mathcal{G}_{exp}$ of graphs $\boldsymbol{G}$ such that each graph occurs in it with probability

$$\mathtt{P}(\boldsymbol{G}) = \frac{1}{Z} \exp\left(-\sum_{i=1}^{s} \beta_i \epsilon_i\right),$$

where Z is the partition function.

Particularly important in the study of complex real-world networks are *small-world* and *scale-free* graphs. Small-world networks were introduced by Watts and Strogatz (1998), whereas scale-free networks were introduced by Barabási and Albert (1999). Intuitively, a small-world graph is one in which each node has "short" paths connecting it to all the other ones. Formally:

Small-world graph

Definition 3.10 {***Small-world graph***} Given a graph $\boldsymbol{G}$ with n nodes, the graph is a *small-world* one if its diameter is $\mathcal{O}(\ln n)$.

Scale-free graph

Definition 3.11 {***Scale-free graph***} Given a graph $\boldsymbol{G}$ with n nodes, the graph is *scale-free* if the distribution of the degrees d of the vertices is governed by a *power law*, namely

Power law distribution

$$p_d \sim d^{-\gamma},$$

for a given γ.

In a scale-free network there are a few nodes with large degrees and many nodes with very small degrees. The power law distribution is explained by the hypothesis of *preferential attachment*: if the network is built up by adding one node at a time, when adding node v_{j+1} the probability that it will be connected to an

Preferential attachment

existing node v_i is proportional to v_i's current degree. The mechanism of preferential attachment is quite old; it was first proposed by Yule (1925) to explain the power-law distribution observed in the number of species per genus of flowering plants. The modern approach to the mechanism was initiated by Simon (1955) in investigating the distribution of the sizes of cities. In the context of complex networks it was re-proposed by Barabási and Albert (1999).

After providing this brief reminder of basic graph notions, we now introduce the first **NP**-complete problem that is central to this book, namely the SAT problem.

3.3 The SAT problem

SAT problem

The name SAT is an abbreviation of *satisfaction*: the SAT problem is concerned with the satisfiability of logical formulas in propositional calculus. The SAT problem was the first to be proved **NP**-complete, in a famous theorem of Cook (1971).

Boolean SAT problems play a crucial role in many important practical applications (Biere *et al.*, 2009). Moreover, such problems are strongly connected with operational research and artificial intelligence. For this reason, the field has been deeply investigated and tremendous progress in SAT solver performance has been observed. Open source implementations are available today that scale up to extremely large problems, with the number of variables and clauses in the order of millions.

Definition 3.12 {***SAT problem***} Let $\mathbf{X} = \{x_1, \ldots, x_n\}$ be a set of n Boolean variables, which can take values in $\{1, 0\}$, where 1 corresponds to *true* and 0 to *false*. Let $\mathbf{\Gamma} = \{\gamma_1, \ldots, \gamma_m\}$ be a set of clauses, i.e., disjunctions of variables or their negations. The SAT problem consists in either finding an assignment of truth values to the variables in $\mathbf{X}$ such that all the clauses in Γ are satisfied or proving that none exists. The conjunction of all clauses in Γ is a formula $\varphi(x_1, \ldots, x_n)$ that must be satisfied. The SAT problem is said to be *satisfiable* if φ can be made true by some choice of the variables, otherwise it is said to be *unsatisfiable*.

EXAMPLE

Let $\mathbf{X} = \{x_1, x_2, x_3, x_4\}$ be a set of four variables, and let $\mathbf{\Gamma}_s = \{\gamma_1, \gamma_2, \gamma_3\}$ and $\mathbf{\Gamma}_u = \{\gamma'_1, \gamma'_2\}$ be two sets of clauses such that

$$\gamma_1 = x_1 \vee \bar{x}_2, \qquad \gamma_2 = x_1 \vee x_3 \vee \bar{x}_4,$$

$$\gamma_3 = \bar{x}_1 \vee x_2 \vee \bar{x}_3,$$

$$\gamma'_1 = x_1, \qquad \gamma'_2 = \bar{x}_1,$$

where $\bar{x}_i$ means the negation of x_i. It is easy to see that the set $\mathbf{\Gamma}_s$ corresponds to a satisfiable problem. In fact $x_1 = 1, x_2 = 1, x_3 = *, x_4 = *$ is a family of solutions. On the contrary, the set $\mathbf{\Gamma}_u$ corresponds to an unsatisfiable problem.

The symbol * denotes any value.

We will now illustrate the relationship between the SAT problem and statistical physics. In order to do this, we introduce a subproblem of SAT, namely the k-SAT problem.

k-SAT problem

Definition 3.13 $\{$***k-SAT problem***$\}$ A k-SAT problem is a SAT problem in which all clauses in $\mathbf{\Gamma}$ have exactly k terms (variables).

EXAMPLE

Let $\mathbf{X} = \{x_1, x_2, x_3\}$ be a set of three variables, and let $\mathbf{\Gamma} = \{\gamma_1, \gamma_2, \gamma_3\}$ be a set of clauses such that

$$\gamma_1 = x_1 \vee \bar{x}_2, \qquad \gamma_2 = \bar{x}_1 \vee x_3,$$

$$\gamma_3 = x_2 \vee \bar{x}_3.$$

This is an example of a 2-SAT problem. The problem is satisfiable, for instance by the triple $\{x_1 = x_2 = x_3 = 1\}$.

It is well known that the 1- and 2-SAT problems are polynomial, whereas the k-SAT problem for $k \geqslant 3$ is $\mathbf{NP}$-complete. The 3-SAT problem is the most studied of the class and is the one for which the most precise results have been obtained.

A generic k-SAT problem is a member of an ensemble generated as described in the following.

Generative model of k-SAT

Definition 3.14 $\{$***Generative model of k-SAT***$\}$ Given two integers k and m and a set $\mathbf{X} = \{x_1, \ldots, x_n\}$ of variables, each clause $\gamma_i \in \mathbf{\Gamma}$ $(1 \leqslant i \leqslant m)$ is generated independently by selecting randomly, with uniform probability, k variables from $\mathbf{X}$; then, each selected variable x_j is negated with probability $p = 0.5$.

Let Π_k be the set of all k-SAT problems with m clauses over n variables. The cardinality of Π_k is

$$|\Pi_k| = \binom{2^k \binom{n}{k}}{m}.$$

Each element of Π_k is a formula $\varphi(x_1, \ldots, x_n)$, i.e., a set of clauses as stated in Definition 3.12; these elements occur in Π_k with equal probability.

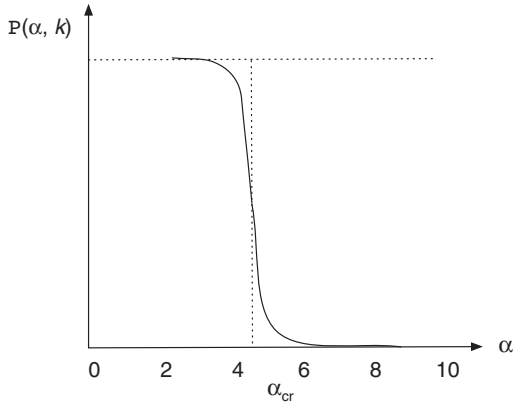


Figure 3.3 Qualitative behavior of the probability $\mathtt{P}(\alpha, k)$ of solution of a random instance of the 3-SAT problem vs. the parameter $\alpha = m/n$.

The generative model for a k-SAT problem deserves further comment. According to Definition 3.14 it may happen that in a given problem instance two clauses turn out to be equal, thus reducing the difficulty of satisfying the problem. In order to avoid this situation the generation of the clauses should be done in another way: first, all the possible k-term clauses are constructed and then m from these are selected without replacement. Even though this model of formula generation allows the probability of any single problem instance to be computed exactly, it is impractical in use because the number of possible clauses is $\binom{n}{k}2^k$. Thus, in practice, in order to generate k-SAT problem instances the procedure of Definition 3.14 is used. To cope with the above difficulty, there are two possibilities: either duplicated clauses are filtered out (which can be done automatically) or the problem is ignored, as the probability of duplication is very low. More precisely, the probability that at least one duplication occurs can be evaluated as follows:

Duplication probability

$$\mathtt{P}_{dupl} = 1 - \frac{\left[2^k \binom{n}{k}\right]!}{\left[2^k \binom{n}{k} - m\right]! \left[2^k \binom{n}{k}\right]^m}. \tag{3.2}$$

For example, in a small problem instance, with, say, $n = 50$, $k = 3$ and $m = 10$, we have

$$\mathtt{P}_{dupl} = 0.000286.$$

The probability of duplication rapidly decreases when the problem size increases.

Experimental investigation of the k-SAT problem has concentrated on the probability $\mathtt{P}(\alpha, k)$ that a randomly chosen formula $\varphi(x_1, \ldots, x_n)$ is satisfiable,

with $\alpha = m/n$. In Figure 3.3 a graph qualitatively describing this probability versus the parameter α is given. From the figure, a remarkable behavior emerges (Monasson, 2002): for all values $\alpha < \alpha_{cr}(k)$ the probability $\mathtt{P}(\alpha, k)$ assumes values very close to 1, whereas when $\alpha > \alpha_{cr}(k)$ it assumes values very close to 0. The *critical value* $\alpha_{cr}(k)$, which is a function of k, defines the boundary between two phases, the SAT phase, where a randomly chosen formula $\varphi(x_1, \ldots, x_n)$ is almost surely satisfiable, and the *UNSAT* phase, where a randomly chosen formula $\varphi(x_1, \ldots, x_n)$ is almost surely unsatisfiable. The random choice of formula means that the way in which k-SAT problem instances (the formulas $\varphi(x_1, \ldots, x_n)$) are generated produces, in the SAT (*UNSAT*) region, very few unsatisfiable (satisfiable) ones, so that a random choice would produce an unsatisfiable (satisfiable) formula with an extremely low probability. In order to find an unsatisfiable (satisfiable) formula in the SAT (*UNSAT*) phase, such a formula must be constructed specifically. This behavior recalls a *phase transition*, as in physical systems. In the vocabulary of phase transitions, the probability $\mathtt{P}(\alpha, k)$ is the *order* parameter and α is the *control* parameter.

Critical value of the control parameter α

2-SAT

The presence of a boundary between the SAT and the *UNSAT* phases has been proved rigorously for the polynomial 2-SAT problem, where the critical value is $\alpha_{cr}(2) = 1$. For the **NP**-complete problem k-SAT with $k \geqslant 3$, the location of the phase transition has not been calculated precisely, but only estimated. For instance, $\alpha_{cr}(3) \simeq 4.3$ and exact lower and upper bounds are known:

$$\alpha_{lb} = 3.26 < \alpha_{cr}(3) < 4.51 = \alpha_{ub}.$$

3-SAT

Using the cavity equations introduced in Section 2.8, Mertens *et al.* (2006) derived threshold values for the parameter α of k-SAT. For $k = 3$ they found $\alpha_{cr} \simeq 4.267$. The authors also provided some closed expressions for these thresholds, for large k. The results of their work support the conjecture that this computation gives the exact satisfiability threshold.

The importance of the critical value is that the most difficult instances are located in its neighborhood. In fact, in correspondence with the phase transition, the computational complexity involved in finding a solution (or in proving that none exists) shows a marked peak, as represented in Figure 3.4. The explanation of this *easy–hard–easy* pattern for the algorithms used to solve a k-SAT problem is that in the SAT phase the problem instances are undersconstrained and there are many solutions, so that it is easy to find one; in the *UNSAT* phase the problem instances are overconstrained and so it is easy to show that no solution exists. In correspondence with phase transitions, there are few solutions for the solvable instances and none for the unsolvable instances; thus, it is difficult to separate SAT from *UNSAT* instances.

The SAT problem can be described in graphical form by means of a bipartite graph, called a *factor graph*.

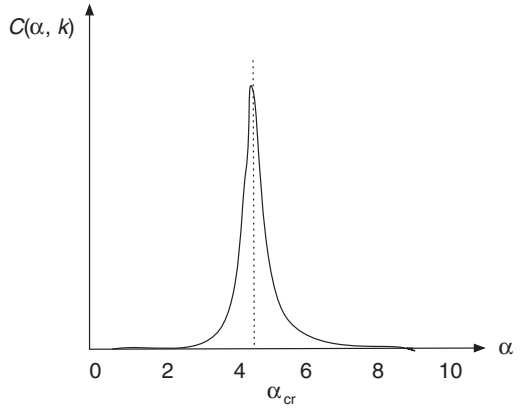


Figure 3.4 Qualitative behavior of the computational complexity $C(\alpha, k)$ for solving a k-SAT problem instance.

Definition 3.15 $\{$***Factor graph***$\}$ Given a SAT problem instance with n variables and m clauses, let us associate with each variable and each clause a node in a *bipartite* graph. Variable nodes are represented as circles, and clause nodes are represented as squares. Edges can connect variable nodes and clause nodes. An edge connecting a clause node γ and a variable node x is labeled with 1, and represented by a continuous line, when x occurs in γ as a *positive* literal; it is labeled with -1 and represented by a dotted line when x occurs *negated*. Variable nodes have a status which can assume three possible values: 1 (true), 0 (false), and $*$ (undecided). Factor graph

An example of such a bipartite graph is provided in Figure 3.5.

3.3.1 SAT problems and the Ising model

We have reached the stage where it is possible to establish a one-to-one correspondence between the SAT problem and the Ising model of spins (Monasson and Zecchina, 1997). In Table 3.1 this correspondence is summarized. The key point in creating this correspondence is to introduce an "energy" function that counts the number of unsatisfied clauses for a given assignment of the variables. Clearly, if the problem instance is *SAT* then this number goes to zero and the corresponding ground state has energy equal to zero; if the problem instance is *UNSAT* then this number remains greater than zero and the corresponding ground state has strictly positive energy. A variable x_j with value *true* Energy function

SAT as an Ising model

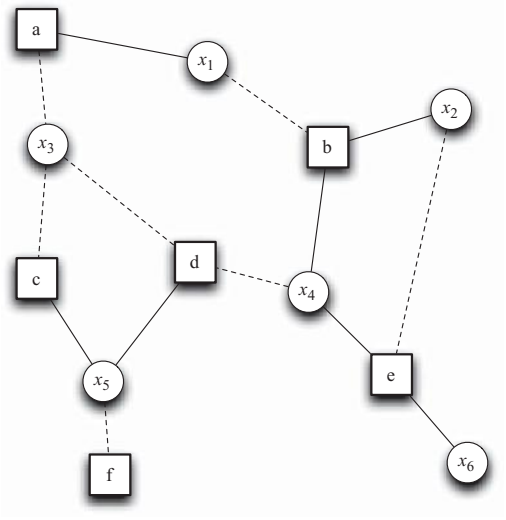


Figure 3.5 Example of a bipartite graph corresponding to the SAT formula $\varphi(x_1, \ldots, x_6) = \gamma_\mathrm{a} \wedge \gamma_\mathrm{b} \wedge \gamma_\mathrm{c} \wedge \gamma_\mathrm{d} \wedge \gamma_\mathrm{e}$, where $\gamma_\mathrm{a} = x_1 \vee \bar{x}_3$, $\gamma_\mathrm{b} = \bar{x}_1 \vee x_2 \vee x_4$, $\gamma_\mathrm{c} = \bar{x}_3 \vee x_5$, $\gamma_\mathrm{d} = \bar{x}_3 \vee \bar{x}_4 \vee x_5$, $\gamma_\mathrm{e} = \bar{x}_2 \vee x_4 \vee x_6$.

corresponds to a spin σ_j with value $+1$, whereas a variable x_j with value *false* corresponds to the spin σ_j with value -1. As exemplified in Table 3.1, given a clause

$$\gamma = x_{j_1} \vee x_{j_2} \cdots \vee x_{j_r} \vee \bar{x}_{h_1} \vee \bar{x}_{h_2} \cdots \vee \bar{x}_{h_s},$$

the associated energy can be written as follows:

$$E(\gamma) = \frac{1}{2^{r+s}} \prod_{i=1}^{r} (1 - x_{j_i}) \prod_{i=1}^{s} (1 + x_{h_i}). \tag{3.3}$$

Random clauses can be encoded in an $m \times n$ matrix $\mathcal{C}$, whose rows correspond to clauses and columns to variables. An entry c_{ij} assumes values in $\{+1, 0, -1\}$. More precisely: $c_{ij} = +1$ if the variable x_j occurs in the clause γ_i as a *positive literal* x_j; $c_{ij} = -1$ if the variable x_j occurs in the clause γ_i as a *negative literal* $\bar{x}_j$; $c_{ij} = 0$ if the variable x_j does not occur in the clause γ_i. The matrix $\mathcal{C}$ encodes the *quenched disorder* of the problem and is a random variable generated as in Definition 3.14. Table 3.2 shows the matrix $\mathcal{C}$ corresponding to the problem represented in Figure 3.5. Given the matrix $\mathcal{C}$, the total energy of the corresponding SAT problem becomes

Quenched disorder

$$E(\mathcal{C}) = \sum_{i=1}^{m} \frac{1}{2^{\sum_{j=1}^{n} |c_{ij}|}} \prod_{j=1}^{n} (1 - c_{ij} x_j). \tag{3.4}$$

Table 3.1 Correspondence between the SAT problem and the Ising model of spins

SAT	Ising model
Boolean variable $x \in \{1, -1\}$	Ising spin $\sigma \in \pm 1$
Clauses	Couplings
Number of clauses violated by a logical configuration	Energy E of the spin configuration
Example	
2-Sat $x \vee \bar{y}$	$E = \frac{1}{4}(1-\sigma_x)(1+\sigma_y)$
$(x \vee \bar{y}) \wedge (\bar{x} \vee z)$	$E = \frac{1}{4}(1-\sigma_x)(1+\sigma_y) + \frac{1}{4}(1+\sigma_x)(1-\sigma_z)$
3-Sat $x \vee \bar{y} \vee z$	$E = \frac{1}{8}(1-\sigma_x)(1+\sigma_y)(1-\sigma_z)$
Minimum number of violated clauses	Ground state energy
Problem is = satisfiable / unsatisfiable	Ground state energy $= 0$ / > 0

Table 3.2 The matrix $\mathcal{C}$ describing the SAT problem reported in Figure 3.5

	x_1	x_2	x_3	x_4	x_5	x_6
γ_a	1	0	−1	0	0	0
γ_b	−1	1	0	1	0	0
γ_c	0	0	−1	0	1	0
γ_d	0	0	−1	−1	1	0
γ_e	0	−1	0	1	0	1

Knowing the probability distribution of the quenched disorder (the matrix $\mathcal{C}$) the average value of the energy can be computed.

For each clause (row in $\mathcal{C}$), we have

$$-\sum_{j=1}^{n} c_{ij} x_j = \text{number of wrong}^{[1]} \text{ literals in clause } \gamma_i. \tag{3.5}$$

In order to explain expression (3.5), let us notice that the product $c_{ij}x_j$ is equal to 1 iff both c_{ij} and x_j have the same sign (i.e., they are both equal to 1 or both equal to -1), it is equal to -1 iff c_{ij} and x_j have different signs (one

[1] This means that c_{ij} and x_j have opposite signs.

equal to -1 and one equal to 1), and it is equal to zero iff $c_{ij} = 0$. If $c_{ij} = 0$, the variable x_j does not occur in clause γ_i and the product $c_{ij}x_j$ does not contribute either to the number of correct literals or to the number of wrong literals. If $c_{ij}x_j = 1$, either the variable x_j is positive in γ_i and so assumes the value 1 (it is true), or it is negative in γ_i and so assumes the value -1 (it is false); in both cases, the value of x_j makes the whole clause γ_i true. On the contrary, if $c_{ij}x_j$ is equal to -1 then the variable x_j assumes a value that makes the corresponding literal false. Thus, if we want to count the number of wrong literals in clause γ_i, we have to sum the products $-c_{ij}x_j$ over j.

Given a clause with k terms, if $-\sum_{j=1}^{n} c_{ij}x_j = k$ then the clause has all literals false and so it is false. For a 3-SAT problem, the condition for which a clause is false reads

$$\sum_{j=1}^{n} c_{ij}x_j + 3 = 0.$$

Then, clause γ_i is not satisfied iff

$$\delta\left(\sum_{j=1}^{n} c_{ij}x_j + 3\right) = 1,$$

where $\delta(z)$ is the Krönecker function, which is 1 if $z = 0$ and 0 otherwise.

Now let $\mathcal{S}$ denote a configuration of spins (variables), i.e., an assignment of truth values to all the n variables. Moreover, let $E(\mathcal{C}, \mathcal{S})$ be the total number of unsatisfied clauses in $\mathcal{S}$:

$$E(\mathcal{C}, \mathcal{S}) = \sum_{i=1}^{m} \delta\left(\sum_{j=1}^{n} c_{ij}x_j + 3\right). \tag{3.6}$$

Ground state energy

The minimum $E_0(\mathcal{C})$ (the ground state energy) can be obtained by the optimal logical assignment to the variables. The value $E_0(\mathcal{C})$ is a random variable, because $\mathcal{C}$ is a random variable; it becomes highly concentrated around the mean value $E_0 = \overline{E_0(\mathcal{C})}$ in the thermodynamic limit ($n \to \infty$). The mean value is computed with respect to the probability distribution of the quenched disorder (namely, the matrix $\mathcal{C}$). Let us now consider the generating function of the energy:

$$G(z, \mathcal{C}) = \sum_{\mathcal{S}} z^{E(\mathcal{C},\mathcal{S})}.$$

The value of E_0 is computed from the average logarithm of $G(z, \mathcal{C})$ (Monasson, 2002), obtaining

$$E_0 = \lim_{z \to 0} \frac{\overline{\ln G(z, \mathcal{C})}}{\ln z}. \tag{3.7}$$

In the SAT region E_0 is equal to 0, whereas it is strictly positive in the *UNSAT* region. The value E_0 as a function of the control parameter α detemines the critical value $\alpha_{cr}(k)$.

EXAMPLE

For the sake of illustration, we follow the treatment of a toy problem, the 1-SAT, provided by Monasson and Zecchina (1997) and Monasson (2002). Let $\mathbf{X} = \{x_1, \ldots, x_n\}$ and $\mathbf{\Gamma} = \{\gamma_1, \ldots, \gamma_m\}$ be the sets of variables and clauses, respectively. In the 1-SAT case the matrix of quenched disorder $\mathcal{C}$, introduced above, has only one entry per row, as all clauses are either a single variable or its negation. Concerning the columns, let us suppose that the clause $\gamma_i = x_j$ occurs in $\mathbf{\Gamma}$. Then, in order to avoid duplication, the same clause cannot occur further in $\mathbf{\Gamma}$. However, the clause $\gamma_h = \bar{x}_j$ may occur in $\mathbf{\Gamma}$ but in this case the problem instance is unsatisfiable, because the formula associated with it contains a variable and its regation, yet both must be true. Thus each column of the matrix $\mathcal{C}$ may have at most two entries different from zero, one equal to $+1$ and one equal to -1.

In order to generate a 1-SAT problem instance we have to extract m elements from the set of possible clauses, which is the union of the set $\mathbf{X}$ and the set containing all the negated variables. We have thus $\binom{2n}{m}$ different problem instances. A unique matrix C is associated with each problem instance, and each matrix can be extracted with equal probability. Thus the probability distribution the quenched disorder is uniform:

$$\mathsf{P}(\mathcal{C}) = \frac{1}{\binom{2n}{m}}. \tag{3.8}$$

For 1-SAT, any matrix $\mathcal{C}$ is described simply by the numbers t_i and f_i of clauses that require a certain variable x_i to be true or false, respectively (Monasson and Zecchina, 1997). Then the partition function is simply

$$Z(\mathcal{C}) = \prod_{i=1}^{n} \left[\exp(-\beta t_i(\mathcal{C})) + \exp(-\beta f_i(\mathcal{C}))\right]. \tag{3.9}$$

By using the replica trick with r replicas, it can be derived that the average of the logarithm of Z, with respect to (3.8), assumes the value

$$\frac{1}{r}\overline{\ln Z(\mathcal{C})} = \ln 2 - \frac{\alpha\beta}{2} + \sum_{h=-\infty}^{\infty} e^{-\alpha} I_h(\alpha) \ln\left(\cosh\frac{\beta h}{2}\right), \tag{3.10}$$

where I_h is the modified Bessel function of order h and $\alpha = m/n$. If expression (3.8) is calculated for vanishing temperature (i.e., for $\beta \to \infty$), the ground state energy density has the value

$$E_0(\alpha) = \frac{\alpha}{2}\left[1 - e^{-\alpha} I_0(\alpha) - e^{-\alpha} I_1(\alpha)\right], \tag{3.11}$$

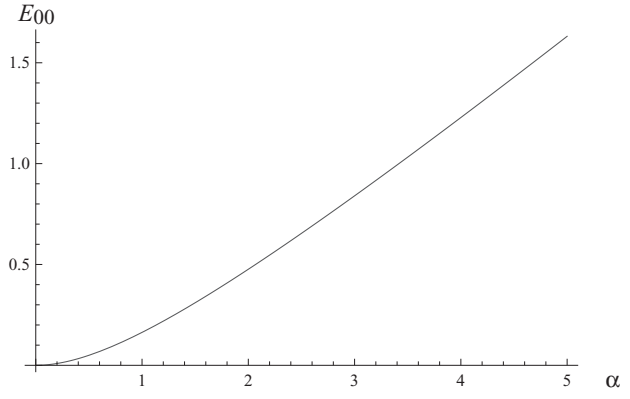


Figure 3.6 The ground state energy E_0 for the 1-SAT problem. The energy is zero only for $\alpha = 0$.

whereas the ground state entropy density is

$$S_0(\alpha) = e^{-\alpha} I_0(\alpha) \ln 2.$$

The graph of the ground state energy for the 1-SAT problem is shown in Figure 3.6. As can be seen, the ground state energy is zero only for $\alpha = 0$. Monasson and Zecchina explain this result by the fact that the entropy is finite and, hence, the number of minima in the energy is exponentially large for any α. In turn this is due to the presence of a fraction $e^{-\alpha} I_0(\alpha)$ of unconstrained variables, whose value does not affect the value of the energy.

3.3.2 Structure of the solution space

An interesting aspect to be investigated is the way in which solutions of k-SAT problems are organized within their respective spaces. If we consider the critical value α_{cr}, then just above this threshold the number of solutions is equal to zero, as we are now in the *UNSAT* phase;[2] however, just below the threshold the typical number of solutions $N_0(\alpha)$ becomes almost surely exponential in the number of variables n (Boufkhad and Dubois, 1999). The quantity that is related to the number of solutions is the *entropy density* $s_0(\alpha)$ of the ground state. In

Entropy density

[2] See Chapter 14 for an alternative view.

fact, each alternative assignment of variables can be considered as a "state", and so

$$s_0(\alpha) = \frac{1}{n} \ln N_0(\alpha).$$

For $\alpha = 0$ we have $s_0 = \ln 2$ because, with a number $m = 0$ of clauses, any variable assignment is a solution of any problem and so $N_0(0) = 2^n$. In order to compute s_0 for any α the replica symmetry method has been applied, but it turns out that this method does not provide a correct solution in the whole *SAT* phase. In fact replica symmetry theory implies that any two assignments (spin configurations, in the statistical physics view) have almost surely the same Hamming distance between them, i.e., they show the same fraction of variables (spins) with a different assignment (orientation). As a consequence, solutions are grouped into a cluster of diameter dn. However, it has been found that this picture is not always true for the whole *SAT* phase (Biroli *et al.*, 2000; Monasson, 2002). More precisely, the *SAT* phase can be divided into two zones by a new critical threshold α_{RSB}:

Transition from one to many solution clusters

- Below α_{RSB} the solution space is replica symmetric (it satisfies the replica symmetry theory), and there is just one cluster of solutions. The Hamming distance d between pairs of solutions is a decreasing function of α.
- At $\alpha_{RSB} \simeq 4.0$ the solutions become grouped into an exponential (in n) number of different clusters, each containing, in turn, an exponential number of solutions. As α increases, the number of clusters decreases. The satisfiability transition corresponds to the vanishing of the clusters.

The threshold α_{RSB} reveals a "clustering phase transition". In Figure 3.7 a qualitative representation of the clustering process is shown. The region $\alpha_{RSB} < \alpha < \alpha_{cr}$ is called the hard-SAT region (Mézard *et al.*, 2005). Here the typical Hamming distance d between solutions in different clusters is about 0.3, and remains almost constant up to α_{cr}. Within each cluster, solutions tend to become more and more similar, with a rapidly decreasing intra-cluster Hamming distance (Monasson, 2002). Given two solutions in the same cluster, it is possible to change one into another by flipping only $\mathcal{O}(1)$ variable values. If the solutions are in different clusters, to change one into the other it is required that a number $\mathcal{O}(n)$ of variables have their values flipped. The clustered region is the most difficult one for many local search algorithms (Semerjian and Monasson, 2003). The presence of clusters of solutions has suggested a new, very effective, algorithm for solving SAT problems, called the survey propagation algorithm. It will be described in Section 3.5.4.

Hard-SAT region

More recently, Montanari *et al.* (2008) refined this picture of the solution space in the *SAT* phase. They have found, for $k \geqslant 4$, a new *condensation* phase

Condensation phase

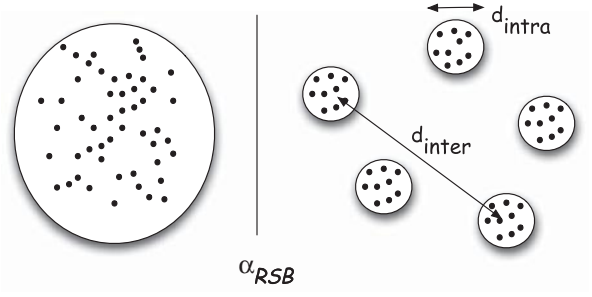


Figure 3.7 Qualitative representation of the clustering process. Below the threshold α_{RSB} solutions are organized into a single cluster, whereas above the threshold an exponential number of clusters, each with an exponential number of solutions, appear.

transition at a value $\alpha_{cd} \in [\alpha_{RSB}, \alpha_{cr}]$. When $\alpha \in [\alpha_{RSB}, \alpha_{cd}]$ there are exponentially many clusters whereas for $\alpha \in [\alpha_{cd}, \alpha_{cr}]$ most solutions are contained in a number of clusters that remains bounded when the number of variables $n \to \infty$. A further refinement of the solution space has been described by Zhou and co-workers (Zhou and Ma, 2009; Zhou, 2010), who have found that, for $k = 3$ and 4, there is a giant connected component that contains different solution groups even when $\alpha < \alpha_{RSB}$. Solutions in the same group are more similar to each other and more densely connected than they are to other solutions.

3.3.3 Backbone

In the *UNSAT* phase the number of solutions is almost surely zero, but another interesting phenomenon appears: in the ground state a *backbone* emerges. A backbone is a set of variables each of which takes on its own specific value in all ground states. It is to be expected that the size of the backbone will be $\mathcal{O}(n)$. Following Monasson (2002), let $\gamma(\alpha, k)$ be the number of these totally constrained variables (the size of the backbone). For $\alpha < \alpha_{cr}$, $\gamma(\alpha, k) = 0$. When the critical value is crossed, two behaviors have been observed: for $k = 2$, $\gamma(\alpha, 2)$ is continuous at the threshold $\alpha_{cr}(2) = 1$ and then increases as the *UNSAT* region is entered further. For $k = 3$ the function $\gamma(\alpha, 3)$ shows a discontinuity, jumping to a finite value γ_{cr} just above the threshold $\alpha_{cr}(3)$. In other words, a finite fraction of variables become suddenly overconstrained when the critical value of α is crossed.

Backbone

Let N_0 be the number of configurations in the ground state of an unsatisfiable formula, i.e., the number of optimal assignments of truth values to the variables.

We can define

$$\mu_i = \frac{1}{N_0} \sum_{a=1}^{N_0} x_i^{(a)}$$

as the average value of the variable x_i over all the ground state configurations. The average μ_i ranges in $[-1, +1]$. When $\mu_i = +1$ ($\mu_i = -1$), the variable x_i is then equal to $+1(-1)$ in all the configurations. If $\mathtt{P}(\mu)$ is the distribution of the values μ_i, the presence of a backbone is denoted by a concentration of the probability values around the values $+1$ and -1. On the contrary, the central region of the distribution ($\mathtt{P}(\mu) = 0$) denotes the presence of variables that are loosely constrained, as their values across the ground state configurations are roughly balanced between $+1$ and -1.

The existence of the backbone has been exploited to design heuristics for solving k-SAT problems. For instance, Dubois and Dequen (2001) showed that the backbone can be exploited to design efficient algorithms for solving hard unsatisfiable 3-SAT formulas (Dequen and Dubois, 2006). Though useful for heuristics, the backbone in itself is **NP**-hard to find. More than that, unless $\mathbf{P} = \mathbf{NP}$, finding a sound approximation to it is also intractable in general (Kilby *et al.*, 2005).

3.3.4 Backdoors

An interesting notion, which parallels that of a backbone and applies to solvable SAT instances, is that of a backdoor. Backdoors were introduced by Williams *et al.* (2003). Informally, a *backdoor* to a given problem is a subset of variables such that, once the variables in this subset are assigned values, the polynomial propagation mechanism of the SAT solver (the "subsolver") solves the remaining formula.[3] Backdoors correspond to clever shortcuts in the solution process. More formally, in the case of satisfiable SAT instances a backdoor is defined as follows.

Definition 3.16 {***Backdoor***} Given a satisfiable SAT problem represented by a formula φ to be satisfied, a *backdoor* to φ with respect to a subsolver $\mathcal{A}$ is a non-empty set of variables S for which there exists an assignment $a_S : S \rightarrow \{0, 1\}^{|S|}$ for which $\mathcal{A}$ returns a satisfying assignment $\varphi(a_S)$.

We may notice that a particular kind of backdoor is provided by a set of *independent* variables. Williams *et al.* (2003) also introduced the notion of *strong backdoor* in the case of possibly unsatisfiable instances.

Definition 3.17 {***Strong backdoor***} Given a SAT problem represented by a formula φ to be satisfied, a *strong backdoor* to φ with respect to a subsolver $\mathcal{A}$

[3] The formal definition of a subsolver can be found in Williams *et al.* (2003).

is a non-empty set of variables S such that, for all assignments $a_S : S \to \{0,1\}^{|S|}$, either $\mathcal{A}$ returns a satisfying assignment or the unsatisfiability of $\varphi(a_S)$ is established.

While setting a backbone to a given value is a *necessary* condition, setting up a (strong) backdoor is a *sufficient* condition for solving a problem. Knowing the backdoor of a SAT problem (i.e., the variable set S) may strongly reduce the complexity of the search. In fact, the exponential part of the search cost is $\mathcal{O}(2^{|S|})$, the rest of the computation being polynomial. The size of a backdoor cannot exceed $n-1$ but the experimental investigation by Williams *et al.* (2003) showed that, in practice, backdoors may be quite small. This implies that, even including in the overall computational cost that for finding a backdoor, a gain in cost may still be the result, in comparison with searches without backdoors. The authors also provided a strategy to exploit backdoors formally.

Finding a backdoor to a problem is intractable in general if $\mathbf{P} \neq \mathbf{NP}$, as mentioned above. However, Kilby *et al.* (2005) showed that the problem becomes tractable, in certain cases (for instance for 2-CNF polynomial subformulas), if the size of the backdoor can be bounded.

3.4 The random $(2+p)$-SAT

An interesting approach to studying the differences in the behavior of 2-SAT and 3-SAT problems was proposed by Monasson *et al.* (1999), who investigated an intermediate problem, called (2 + p)-SAT, which is a SAT problem with clauses containing either two or three literals. More precisely, any problem instance contains pm clauses with three literals and $(1-p)m$ clauses with two literals. The problem is still $\mathbf{NP}$-complete because any instance contains, for $p > 0$, a subformula with at least three clauses. The authors aimed at computing the threshold $\alpha_{cr}(2+p)$ for fixed p, knowing that $\alpha_{cr}(2) = 1$ and $\alpha_{cr}(3) \simeq 4.267$. By observing that 2-SAT problem instances (or 3-SAT problem instances) are almost always unsatisfiable for some number of clauses n (or $\alpha_{cr} n$), the following relation holds:

$$\alpha_{cr}(2+p) \leqslant \min\left(\frac{1}{1-p}, \frac{\alpha_{cr}(3)}{p}\right).$$

Let us now consider the probability distribution $\mathsf{P}(\mu)$ introduced in Section 3.3.3 and analyze what happens at the threshold of the 2-SAT and 3-SAT cases. Let $f(k,\alpha)$ be the fraction of variables that become totally constrained at and above the threshold. The function $f(k,\alpha)$ is identically zero below the threshold for both cases. For 2-SAT, the function becomes strictly positive above $\alpha_{cr}(2) = 1$ but is continuous at the transition point, i.e., $f(2,1^+) = f(2,1^-) = 0$. On the

contrary, for 3-SAT the function $f(3, \alpha)$ shows a discontinuity, i.e., $f(3, \alpha_{cr}^{-}) = 0$ and $f(3, \alpha_{cr}^{+}) = f_{cr}(3) > 0$.

For the mixed $(2 + p)$-SAT model, the key point is understanding how a discontinuous 3-SAT-like transition may appear when the parameter p is increased from 0 to 1. By applying a previously introduced method (Monasson and Zecchina, 1997), Monasson *et al.* (1999) found that the model has a continuous *SAT–UNSAT* transition below the value $p_c = 0.413$ at an α_{cr} value given by

$$\alpha_{cr}(2 + p) = \frac{1}{1 - p}. \tag{3.12}$$

2-SAT versus 3-SAT

Then, below the value $\alpha = 1/(1 - p)$ the $(2 + p)$-SAT problem behaves as 2-SAT. For $p > p_c$ the transition becomes discontinuous and the model behaves as 3-SAT.

Achlioptas *et al.* (2001b) studied the $(2 + p)$-SAT problem in the limit $n \to \infty$ and found, by combinatorial arguments, that the critical value for p is $p_c = 2/5$. In response to this new result, Biroli *et al.* (2000) proved by statistical physics methods (though not rigorously) that p_c indeed equals $2/5$.

3.5 Solving the SAT problem

In this section we will briefly review the different approaches used to solve SAT problems. More specifically, we will be interested in discussing their behavior when problem instances get close to the phase transition region.

"Regular" problem instances versus random ones, where no regularities may be assumed to exist

The basic assumption underlying the heuristics exploited by many SAT solvers is that the problem instances occurring in practice are usually easy even when their size is large. This happens because these instances frequently exhibit some kind of regularity, which allows specific heuristics to be exploited. However, problems inside the phase transition region are not of this kind and then the heuristics fail.

A first family of SAT solvers contains *exact* (or *complete*) solvers. They have been designed to always find a solution when one exists. Nevertheless, in order to prevent their running for an undetermined time period, they are also provided with a time-out mechanism. Therefore, they classify a problem as unsatisfiable either when they prove that a solution does not exist or when they run out of time.

A second family, developed for practical applications to engineering problems, is that of *incomplete* SAT solvers. They use heuristics that are easy to apply, such as hill climbing, but are incomplete, i.e., large regions of the search space are disregarded and remain unexplored. If the solution is in such a region, it will not be found; thus a solvable problem instance is declared unsolvable.

MaxSAT problem

A third family addresses a generalization of SAT called MaxSAT. Given a Boolean expression in conjunctive normal form (CNF), a MaxSAT problem consists in finding a variable assignment satisfying the maximum possible number of clauses, i.e., minimizing the number of violated constraints. MaxSAT reduces to SAT when the problem instance is satisfiable. It has a strong relevance for practical applications and thus several MaxSAT solvers have been developed. It should be noted that, as we will discuss later, the SAT solvers mentioned above are not able to deal with MaxSAT.

Finally, we will give special attention to a new family of algorithms that were designed for solving random SAT problems located close to the phase transition region.

3.5.1 Exact SAT solvers

DPLL algorithm

Most exact SAT solvers are based on the well-known *Davis–Putnam–Logemann–Loveland* (DPLL) algorithm (Davis and Putnam, 1960; Davis *et al.*, 1962) and include some smart heuristics to avoid the useless exploration of regions of the solution space where solutions are impossible. This is a complete, backtrack, algorithm; if the heuristic is complete, the SAT solver will find a solution. Usually, the solution first found is returned. There are two key strategies, which make modern SAT solvers based on DPLL very effective: (i) *fast unit propagation* and (ii) *clause learning* (Marques-Silva and Sakallah, 1996, 1999; Moskewicz *et al.*, 2001; Eén and Sörensson, 2004; Sörensson and Eén, 2005).

The search strategy of DPLL is *conflict* driven. It starts with a tentative hypothesis about a possible assignment of the Boolean variables and then checks every single clause. When a conflict is detected, i.e., a clause is not satisfied by the current hypothesis, the algorithm backtracks, modifies the hypothesis, and resumes the computation. The initial hypothesis contains only one variable; then the hypothesis is extended further to include new variables until all the variables have been assigned. Unit clauses are those that are not satisfied by the current hypothesis and have only one more variable to be assigned. Therefore, they set a deterministic constraint on the value of the unassigned variable. The fast propagation of unit clause constraints focuses the search dramatically (Moskewicz *et al.*, 2001).

Fast unit-propagation heuristics

Clause learning

Clause learning is activated when the current hypothesis cannot be completed because it is impossible to find a consistent assignment for the remaining variables. In order to prevent the algorithm from generating another hypothesis entailing the same contradiction, a clause characterizing the conflicting variable assignment is learned (Marques-Silva and Sakallah, 1996). Such clauses are called nogoods. Nogoods are used by a constraint propagation algorithm to restrict the space of assignments that can be generated.

Nogoods

The most popular SAT solver of this family is MiniSat (Sörensson and Eén, 2005), which is available in an open-source package; it can be customized depending on the needs of the application.

An alternative to conflict-driven approaches is look-ahead (Heule *et al.*, 2004). Both unit propagation and clause learning can be combined with look-ahead, leading to more complex implementations. Look-ahead solvers are usually stronger on hard problem instances while conflict-driven ones are faster on large, but easy, problem instances.

Look-ahead

3.5.2 Incomplete SAT solvers

Incomplete SAT solvers use local search strategies, typically hill-climbing or gradient descent, which explore narrow regions of the hypothesis space. Unit clause propagation and other heuristics used by complete solvers can be exploited by incomplete solvers as well (Schuurmans and Southey, 2001; Hirsch and Kojevnikov, 2005). However, in order to escape the local minima inherent in local search, incomplete algorithms include a *stochastic restart*. The idea consists in choosing another region of the assignment space, not yet explored, and then repeating another phase of local search. From the theoretical point of view this strategy reintroduces completeness, in the limit. The many algorithms of this family differ in their heuristics and in the stochastic components used for deciding the restart point in the space of possible assignments. Solvers based on genetic algorithms, simulated annealing, and random walk have also been proposed.

Stochastic restart

Among the most popular SAT solvers of this family WalkSAT (Kautz and Selman, 1996), GSAT (Selman *et al.*, 1992) and UnitWalk (Hirsch and Kojevnikov, 2005) are worth mentioning. The field is being actively investigated because this approach seems the most promising for solving hard problems, which are out of the reach of exact solvers.

3.5.3 MaxSAT solvers

MaxSAT solvers do not involve the assumption that the given problem instance is solvable; they are designed to find the maximally consistent subset of the constraints (clauses). This prevents MaxSAT solvers from using the most effective strategies exploited by SAT solvers, i.e., unit propagation and clause learning. In fact, a maximum satisfiability solution can be one that violates a unit clause and so unit propagation would set a misleading bias on the search space. In an analogous way clause learning is activated when a conflict arises, in order to set a constraint that prevents parts of the search space being explored again. In this

case, ignoring the conflict, dropping the violated constraints, and continuing the search might be the best choice.

For the above reasons, progress in the area of MaxSAT has been much slower than for SAT and the available solvers are able to deal only with problem instances of a size much smaller than those solved by SAT.

As a matter of fact, MaxSAT solvers are adaptations of the SAT solvers described in the previous subsections. One group is again based on the DPLL algorithm (Davis *et al.*, 1962) but does not make use of unit propagation and clause learning. Currently, the best representatives of this approach are MaxSATZ (Li *et al.*, 2007a) and MSUF (Marques-Silva and Manquinho, 2008), whose performances are orders of magnitude lower than those of SAT solvers such as MiniSAT (Sörensson and Eén, 2005). A second group is based on local search and multiple restarts. Actually, local search seems to be more suitable for this kind of problem. Even if such a method is incomplete, it can find better approximations for large problem instances than exhaustive search does. A typical heuristic used to guide local search is the gradient of the number of satisfied clauses, which implicitly accepts violated clauses.

State-of-the-art MaxSAT solvers based on local search are SAPS (Hutter *et al.*, 2002) and ADAPTG2WSAT (Li *et al.*, 2007b). Finally, recent progress has been made by Kroc and his colleagues, whose paper (Kroc *et al.*, 2009) informed the brief review presented here.

3.5.4 Survey propagation

The *survey propagation* algorithm (denoted SP in the following) is a SAT solver based on heuristics completely different from those discussed so far (Braunstein *et al.*, 2008). It was designed with the purpose of trying to solve instances close to the phase transition region, where the solutions are grouped into clusters (see Section 3.3.2).

The basic ideas come from statistical physics, i.e., the *cavity method* (see Section 2.8) and the dynamics of *spin glasses*. However, a strict analogy between the basic mechanisms of the algorithm and popular methods, such as belief propagation, in artificial intelligence was acknowledged later.

In its essence, SP is still a backtrack algorithm that tries to find a consistent solution. However, it uses powerful heuristics, so that it quite rarely needs to backtrack. The basic strategy consists in trying to identify the solution clusters, called *covers*. Then a specific solution is searched for inside each cluster using a local search strategy, but discovering solution clusters is *per se* a task even more complex than discovering single solutions. Survey propagation circumvents the problem by computing a statistical approximation to the covers; thus the covers are just probabilistic guesses, which are much less expensive to

Factor graph

compute than true solution clusters. The procedure uses the *factor graph* representation of the SAT problem introduced in Definition 3.15 (Frey *et al.*, 1998; Kschischang *et al.*, 2001) and exemplified in Figure 3.5. The nodes in the factor graph interact iteratively, exchanging signals along the edges until a stationary status is reached.

Nodes in the factor graph interact through warnings and requests.

The graph nodes exchange two kinds of signals: (i) *warnings* and (ii) *requests*. Warnings flow from variable nodes to clause nodes and report the status of the variables, whereas requests flow from clauses to variables. When a clause γ *needs* a specific status for a variable x, because it is not satisfied with the status of the other variables to which it is connected, it sends a request to x asking for a status change. A variable decides its status on the basis of the requests from the clauses to which it is connected, using a majority-voting policy, and then updates the warning signal. In principle, one can think of simulating the behavior of the network and observing what then happens. The algorithm for doing this was supplied by Braunstein *et al.* (2008) as a preliminary introduction to SP and has been called *warning propagation* (WP). Actually, WP models the *undecided* status. The nodes evolve, changing their status, until the network either converges to a stable state or does not.

Starting from this basic model there are methods for extracting a problem solution. However, SP goes beyond it. Instead of explicitly modeling a system's evolution, it estimates the probability for a variable to reach a specific status. Here the *undecided* status is introduced to model the fact that a variable can float between 0 and 1. To this end the factor graph is extended by labeling the edges with real values in the interval $[0, 1]$, representing the probabilities of occurrence of the signal values (request and warning) flowing through them. In an analogous way, variable nodes have associated probabilities. The algorithm for estimating probabilities is based on the cavity method and bears a strong resemblance to the belief propagation method used in Bayesian networks (Pearl, 1988). Here the cavity method, originally introduced for infinite systems, is adapted to handle factor graphs, obtaining the *cavity graph* introduced in Section 2.8 and exemplified in Figure 2.4(b).

Belief propagation

Cavity graph

The probabilities of the three possible values $\{0, 1, *\}$ of a variable node x_i are computed on the basis of the probabilities of the values of the requests that x_i will receive from the clause nodes to which it is connected. In turn, the latter probabilities are computed on the basis of the node status and the warning probabilities received by the clause nodes from the variable nodes to which they are connected. The probabilities of the values received by x_i are computed using the cavity graph $\boldsymbol{G}$. Implicitly, this is equivalent to making an independence assumption, that the probability values of the signals received by x_i from its neighbors do not depend on x_i itself. This is true only in the case where $\boldsymbol{G}$ is a tree. In the general case an approximate estimate is the result.

Values with higher probability are tried first in the local search phase

The probability estimation algorithm is embedded in an iterative procedure, which is run until convergence to a stable distribution is obtained. Then solution clusters are identified. A solution cluster (cover) is an assignment where some variable nodes have the value $*$ with high probability. In fact, the cluster extraction algorithm estimates the probability of the most likely cover. Then, starting from this cover, a local search procedure (called *decimation*) is activated; it tries to find a solution, assigning a Boolean value 0 or 1 to the undecided variables in the cover. The probability assigned to the values 0 and 1 in the previous step is used as a heuristic for guiding the search. A clear description of the SP algorithm was provided by Braunstein *et al.* (2008).

3.6 Comments

In this chapter we have indicated that surprising links exist between problems in statistical physics and in combinatorial search (in this chapter, the SAT problem) and that synergy between the two fields can bring benefits to both. Beyond the formal translation of an Ising model to a SAT, one may wonder what (if any) are the deep reasons underlying the links. In fact, the problems handled in statistical physics typically involve a large number of interacting components (particles), whose ensemble behavior generates macroscopic properties that we can observe and measure. In SAT the microscopic and macroscopic levels are not readily distinguished, nor are their natures immediately apparent. Intuitively, the macroscopic level consists of two "phases", *SAT* and *UNSAT*, whereas the microscopic level is composed of the variables, which interact with each other through the clauses (the "couplings") that constrain their possible value assignments.

The pattern of interactions is represented by a factor graph, introduced in Definition 3.15. Assuming this interpretation, as only variables occurring in the same clause interact the range of the interaction is determined by the number of terms in the clauses; for instance, in a 2-SAT problem, variables interact only in pairs and we say that this is a case of a "short-range" interaction, whereas a k-SAT problem with large k may be a case where there are "long-range" interactions. A situation in which every pair of variables interacts corresponds to clauses involving all the variables (n-SAT). Obviously, variables occurring in different clauses are nevertheless correlated even though not directly. Augmenting k, while keeping constant the number n of variables, makes the problem easier to satisfy; the critical value α_{cr} scales as $2^k \ln 2$. In this whole picture a central role is played by randomness. In fact, by constructing specific SAT problems, both solvable and unsolvable problems can be obtained for any value of k and of the ratio $\alpha = m/n$. For instance, to build a solvable instance it is sufficient that all clauses include a given variable x_j or that all clauses include its negation

$\bar{x_j}$. Conversely, any problem including the two clauses $r_i = x_j$ and $r_2 = \bar{x}_j$ is unsolvable. From this observation we may conclude that the generative model of SAT problems establishes the syntax of the instances and that, for each k, m, and n, the proportion of satisfiable (unsatisfiable) instances with the above format in the *SAT* (*UNSAT*) region is overwhelming. Thus, in order to precisely pick a satisfiable or unsatisfiable problem instance at will, extra knowledge is necessary.

4

Constraint satisfaction problems

Contents

An important class of $\mathbf{NP}$-complete problems is that of *constraint satisfaction problems* (CSPs), which have been widely investigated and where a phase transition has been found to occur (Williams and Hogg, 1994; Smith and Dyer, 1996; Prosser, 1996). Constraint satisfaction problems are the analogue of SAT problems in first-order logic; actually, any *finite* CSP instance can be transformed into a SAT problem in an automatic way, as will be described in Section 8.4.

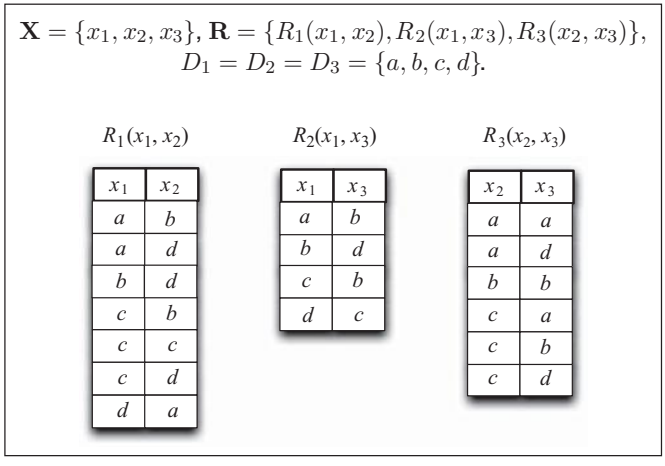
CSP

Formally, a finite CSP is a triple $(\mathbf{X}, \mathbf{R}, \mathbf{D})$. Here $\mathbf{X} = \{x_i | 1 \leq i \leq n\}$ is a set of variables and $\mathbf{R} = \{R_h, 1 \leq h \leq m\}$ is a set of relations, each defining a *constraint* on a subset of variables in $\mathbf{X}$; $\mathbf{D} = \{D_i | 1 \leq i \leq n\}$ is a set of variable domains D_i such that every variable x_i takes values only in the D_i, whose cardinality $|D_i|$ equals d_i. The constraint satisfaction problem consists in finding an assignment in D_i for each variable $x_i \in \mathbf{X}$ that satisfies all relations in $\mathbf{R}$.

In principle a relation R_h may involve any proper or improper subset of $\mathbf{X}$. Nevertheless, most authors restrict investigation to *binary constraints*, defined as relations over two variables only. This restriction does not affect the generality of the results that can be obtained because any relation of arity higher than two can always be transformed into an equivalent conjunction of binary relations.

Tabular representation

A relation R_h involving the variables $x_{i_1}, \ldots, x_{i_j}$ can be represented as a table in which every row contains an admissible assignment $(a_{i_1}, \ldots, a_{i_j})$



x_1	x_2
a	b
a	d
b	d
c	b
c	c
c	d
d	a

x_1	x_3
a	b
b	d
c	b
d	c

x_2	x_3
a	a
a	d
b	b
c	a
c	b
c	d

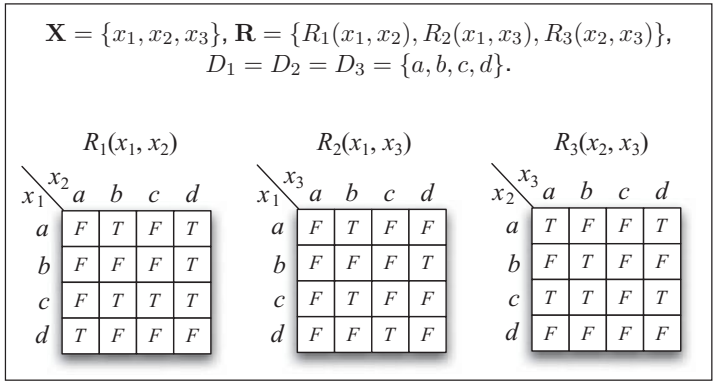
Figure 4.1 Example of a binary CSP where the constraints are represented in tabular form.

to $(x_{i_1}, \ldots, x_{i_j})$ of constants from $D_{i_1} \times \cdots \times D_{i_j}$. Some examples of relations are provided in Figure 4.1. Checking a constraint R_h on a variable assignment $(a_{i_1}, \ldots, a_{i_j})$ to $(x_{i_1}, \ldots, x_{i_j})$ reduces to checking whether the tuple $(a_{i_1}, \ldots, a_{i_j})$ is present in the corresponding table.

If all relations are binary, an alternative representation for the constraints becomes possible. In fact, a binary relation $R_h(x_i, x_j)$ can be represented as a Boolean matrix $\mathbf{M}_{i,j}$ of size $d_i \times d_j$, where an entry contains T (*true*) if the corresponding tuple is admissible and F (*false*) otherwise. Entries containing T are usually called *goods* whereas entries containing F are called *nogoods*. The advantage of this representation is that a constraint can be checked in constant time by means of a single inspection of the matrix. The disadvantage is that the matrices can become very large when the size of the domains grows. Examples of matrix representations are given in Figure 4.2.

Constraint satisfaction problems can be described in some logic language which, most frequently, is chosen to be a subset of first-order logic. Finite CSPs can be described in a function-free first-order logic called DATALOG, typically used in relational learning. We will come back to this point in Chapters 6 and 9; here we will merely introduce the simple transformations necessary to express a finite CSP in DATALOG. DATALOG

Relations and variable domains can be immediately associated with predicates in DATALOG. More specifically, a relation $R_h(x_{i_1}, \ldots, x_{i_j})$ of arity j can be associated with a predicate $p_h(x_{i_1}, \ldots, x_{i_j})$, of arity j having the same variables as arguments; a domain D_i corresponding to variable x_i will be associated



$R_1(x_1, x_2)$	x_2 = a	b	c	d
x_1 = a	F	T	F	T
b	F	F	F	T
c	F	T	T	T
d	T	F	F	F

$R_2(x_1, x_3)$	x_3 = a	b	c	d
x_1 = a	F	T	F	F
b	F	F	F	T
c	F	T	F	F
d	F	F	T	F

$R_3(x_2, x_3)$	x_3 = a	b	c	d
x_2 = a	T	F	F	T
b	F	T	F	F
c	T	T	F	T
d	F	F	F	F

Figure 4.2 Examples of the same binary relations as those in Figure 4.1. Here the constraints are represented as Boolean matrices.

with a unary predicate $u_i(x_i)$. Relations and domains define the semantic interpretation of the corresponding predicates. Moreover, we notice that, in a CSP, the set of constraints defined by the relations in $\mathbf{R}$ must be satisfied simultaneously. This corresponds to a logical *AND*. Therefore any CSP $(\mathbf{X}, \mathbf{R}, \mathbf{D})$ can be translated into a DATALOG formula having the format

$$CSP(x_1, \ldots, x_n) = \bigwedge_{i=1}^{n} u_i(x_i) \bigwedge_{h=1}^{m} p_h(x_{i_1}, \ldots, x_{i_{j_h}}). \tag{4.1}$$

For the example in Figure 4.1, expression (4.1) becomes

$$\begin{aligned} &CSP(x_1, x_2, x_3) \\ &\quad = u_1(x_1) \wedge u_2(x_2) \wedge u_3(x_3) \wedge p_1(x_1, x_2) \wedge p_2(x_1, x_3) \wedge p_3(x_2, x_3). \end{aligned}$$

Therefore, solving the CSP defined by this expression means finding a set of values (substitutions) for the variables x_1, x_2, x_3 that verifies $CSP(x_1, x_2, x_3)$ in the universe defined by $\mathbf{D}$ and $\mathbf{R}$. Artificial intelligence, operational research, and logic programming offer a number of algorithms potentially eligible for this task. However, depending on the specific case, not all algorithms are equivalent from the performance point of view. Globally, the task remains non-polynomial (unless $\mathbf{P} = \mathbf{NP}$). Nevertheless, subclasses of CSPs have been identified that can be described in restricted fragments of DATALOG and can be solved in polynomial time using proper algorithms.

For binary CSPs, a graphical representation can also be used; it consists of an undirected graph $\mathbf{G} = (\mathbf{V}, \mathbf{E})$, with $|\mathbf{V}| = n$ and $|\mathbf{E}| = m'$, called the

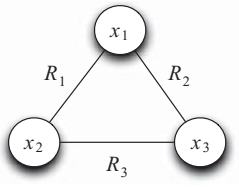


Figure 4.3 Graphical representation of the binary CSP described in Figures 4.1 and 4.2. In this case the graph is completely connected, but this is not typical.

constraint graph. A simple example is provided in Figure 4.3. In the graph $\boldsymbol{G}$, the vertices correspond to variables and the edges correspond to binary relations (constraints). More precisely, an edge connects node x_i with node x_j iff there exists a relation $R_h(x_i, x_j)$ in $\mathbf{R}$ whose arguments are the variables x_i and x_j. If several relations are allowed to share their arguments then the number m' of edges in $\mathbf{G}$ will be smaller than the number m of relations. If this is not allowed then $m = m'$. Notice that the possible presence of unary relations is ignored when considering the structure of the graph. They may possibly be associated with its nodes.

Constraint graph

The aspect of CSPs that primarily interests us, however, is the emergence of a phase transition in sets of randomly generated problem instances. The appearance of a phase transition has been experimentally investigated by several authors, mostly in binary CSPs (Prosser, 1996; Smith and Dyer, 1996; Giordana and Saitta, 2000; Smith, 2001; Xu and Li, 2000). In the following sections we will briefly recall the main results, and in later chapters the relationship between CSPs and learning tasks will be illustrated.

Experimental investigation of phase transition

4.1 Algorithms for solving CSPs

Constraint satisfaction problems encompass a broad class that includes several subclasses, each characterized by peculiar features. This has led to different approaches to find a solution.

A first important subdivision is between "continuous" CSPs and "discrete" CSPs. Continuous CSPs, where constraints usually take the form of inequalities, are typically investigated in operational research and are solved with optimization techniques such as, for instance, linear programming. Symbolic CSPs are usually solved using search algorithms developed for artificial intelligence and logic programming.

Continuous and discrete CSPs

In the following we will briefly review algorithms for solving symbolic CSPs, specifically those that can be described in DATALOG. Not surprisingly, we will see that most methods we described for solving SAT problems have a

correspondence in the more general framework of CSPs. However, the values of variables in CSPs are not simply *true* or *false*, as in SAT, but are selected from their corresponding domains of discrete values.

4.1.1 Generate and test with backtracking

Backtrack search is a general algorithm for solving any kind of search problem, and it represents the baseline for solving a CSP. However, the well-defined formal structure of a CSP allows task-specific heuristics to be exploited that apply to the whole class and make the search much more efficient. Good introductions to CSPs and CSP solvers are provided by Kumar (1992) and Russell and Norvig (2003).

The minimum number of colors required for solving this CSP is three.

For illustrating these algorithms we will use a classical graph coloring problem as a guide line. Thus, suppose that the problem is to paint the faces of a cube in such a way that adjacent faces always have different colors. Suppose, moreover, that three colors are available: red (R), white (W), and green (G).

EXAMPLE

Problem: Assign colors to the six faces of a cube as above.
Available colors: R, W, G.
CSP formulation: Satisfy the logical expression

$$\varphi(x_1, x_2, x_3, x_4, x_5, x_6) = Color(1, x_1) \wedge Color(2, x_2) \wedge Color(3, x_3) \wedge Color(4, x_4) \wedge Color(5, x_5) \wedge Color(6, x_6)$$

under the constraints

$$\begin{aligned} &x_1 \neq x_2 \wedge x_1 \neq x_3 \wedge x_1 \neq x_4 \wedge x_1 \neq x_5, \\ &x_2 \neq x_3 \wedge x_2 \neq x_5 \wedge x_2 \neq x_6, \\ &x_3 \neq x_4 \wedge x_3 \neq x_6, \\ &x_4 \neq x_5 \wedge x_4 \neq x_6, \\ &x_5 \neq x_6. \end{aligned}$$

In the above formulas the predicate $Color(k, x_k)$ has the meaning "Face k has color x_k". The constraint graph is shown in Figure 4.4(a) and the search tree generated by the basic backtracking algorithm is given in Figure 4.4(b). Integers denote the cube faces, whereas the lower-case letters refer to the nodes of the search tree.

The basic search algorithm is very simple. The problem variables $\{x_1, \ldots, x_6\}$ and the set of values characterizing the domain of each variable, i.e., $D =$ (R, W, G), are put into a chosen order *a priori*. Following this assigned order, the algorithm tries to find a value, compatible with the problem constraints, for each

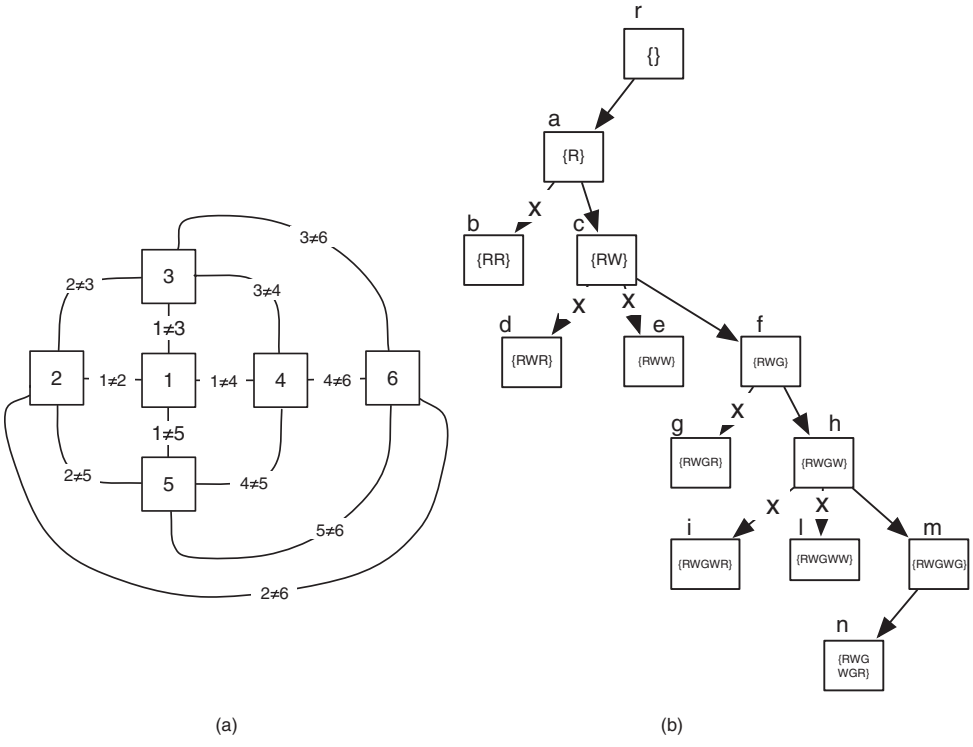


Figure 4.4 Constraint satisfaction problem consisting in coloring the faces of a cube using three different colors, R, W, G, in such a way that adjacent faces cannot have the same color. (a) The constraint graph. (b) A search tree of pure backtracking. Edges tagged with "x" correspond to backtrack actions.

variable. Every time a new assignment is made, the current partial solution is checked against the constraints. If one (or more) is violated, the algorithm backtracks to the previous node and tries again with the next assignment. If a solution exists, the algorithm is guaranteed to find it. Otherwise it will test all the possible assignments before reporting failure. This basic algorithm can be made much more efficient by exploiting heuristics that are applicable to any CSP (Haralick and Elliott, 1980). The following are popular heuristics.

Heuristics

- *Variable and value reordering* In general, selecting variables and values according to an *a priori* order is not the most effective strategy. In fact, the risk of backtracking in subsequent steps can be significantly reduced by selecting the next action on the basis of results obtained in the previous steps. A frequently used and easy to implement strategy consists in giving the preference to the most constrained variables. In our graph-coloring

Variable and value reordering

example this strategy reduces to selecting the cube face whose set of already colored neighborhoods is the largest.

- *Look-ahead* The basic algorithm first expands the tree and then checks for consistency. Many wrong choices and the consequent backtracks can be avoided by selecting values that are compatible with those already assigned. This can be done by introducing a look-ahead step, which checks the constraints before expanding the tree (Frost and Dechter, 1995). Considering the tree in Figure 4.4(b), according to this strategy the nodes d, e, g, i, l, n will not be generated.

Look-ahead

- *Intelligent backtracking* The backtracking strategy used by the basic algorithm, previously described, is called *chronological* backtrack, i.e., the algorithm goes back to the last choice made. It goes back further only when all the choices in the current subtree have failed. An alternative strategy, developed for CSPs, consists in backtracking not to the last choice point but to an upper-level node, where the search looks more promising, thus avoiding a sequence of choices with consequent backtracks that can be predicted to be unsuccessful. This strategy is called intelligent backtracking or *back-jumping* (Gupta, 2000; Prosser, 1995).

Intelligent backtracking

4.1.2 Constraint propagation

Constraint propagation

The most powerful task-specific technique for increasing the efficiency of CSP solvers is *constraint propagation* (Tsang, 1993; Caramia and Dell'Olmo, 2002). When a value is assigned to a variable it is propagated through the constraint graph, removing from the domains of the other variables values that are incompatible with the assigned value. In this way, in subsequent steps choices cannot be made that are incompatible with the values already assigned. Of course, it may happen that the domain of a variable becomes empty, requiring the sequence of assignments already made to be reconsidered.

In the example of Figure 4.4, propagating the first value (R) assigned to variable x_1 (face 1) through the constraint graph has the effect of eliminating R from the domains of the cube faces adjacent to face 1. Thus the domains of x_2, x_3, x_4, and x_5 will contain only the values W and G. After the color of x_2 has been selected as W, the domains of x_3 and x_5 will contain only G, that of x_4 still contains W and G, and that of x_6 contains R and G. So, in this simple case, after a few steps the procedure reduces to a deterministic choice.

In general, constraint propagation is combined with a backtracking algorithm and has the effect of reducing the need for backtracking to the case where one or more variables have an empty domain. Notice that, by limiting constraint propagation to the relations directly affecting a variable in the constraint graph, this

strategy corresponds to a look-ahead step. However, propagation through the full graph allows long-range interactions between variables to be captured, and this leads to the detection of constraint violations. A good strategy, usually adopted in CSP solvers based on constraint propagation, is to select first the most constrained variables.

The most constrained variables should be selected first.

Finally, the constraint propagation technique can be combined with the backtracking heuristics previously mentioned, leading to a family of hybrid search algorithms where techniques such as back-jumping can be made smarter by exploiting the constraint graph (see, for instance, Prosser, 1993).

4.1.3 Local search

As we have already seen for SAT, local search algorithms are quite popular and effective for general CSPs also.

Usually, local search starts with a tentative assignment, which may be chosen randomly or according to some problem-specific heuristic. Then the assignment is progressively refined by the elimination of constraint violations as far as possible. The search process often follows a hill-climbing strategy, selecting transformations that minimize the number of violated constraints. When the search becomes entrapped in a local minimum, from where a consistent solution cannot be reached, a restart is made by selecting another initial hypothesis. Therefore, local search is frequently used in conjunction with stochastic algorithms for selecting the restart points. Moreover, local search for CSPs has been widely exploited in connection with evolutionary algorithms (Michalewicz, 1996) and simulated annealing (Kirkpatrick *et al.*, 1983).

Stochastic algorithms and restart

One popular algorithm, which we have already mentioned for SAT at the end of Section 3.5.2 and which works very well for CSPs, is WalkSAT (Erenrich and Selman, 2003). Another algorithm, which is simple to implement and has performed extremely well on a large number of CSPs, is ChainSAT (Russell and Norvig, 2003; Alava *et al.*, 2008).

4.1.4 MaxCSP

In the previous chapter we saw a generalization of SAT called MaxSAT, which finds a partial solution minimizing the number of violated clauses. In an analogous way, MaxCSP finds a solution satisfying the largest possible subset of constraints. However, in view of the wider framework set by CSPs, MaxCSP includes a variety of subcases, which differ both in the problem setting and in the specific techniques developed for addressing the task.

As many clauses as possible are satisfied.

A first remark is that most MaxCSP research has been done in the area of continuous-valued CSPs and involves linear programming. Here, constraints

come in the form of inequalities such as $x_i \leq x_j$. As we have already mentioned, investigating CSPs in continuous domains is beyond the scope of this book. Nevertheless, it is worth noticing that this kind of CSP is very important for approaches to machine learning based on regularization theory and linear algebra, such as kernel machines (Shawe-Taylor and Cristianini, 2004). Nevertheless, in many cases a continuous CSP can be approximated by a discrete one (Deineko *et al.*, 2008).

Hard and soft constraints

In MaxCSP a fundamental distinction is made between hard constraints and soft constraints. Hard constraints cannot be violated whereas soft constraints provide options to be optimized. In other words, a solution to a MaxCSP must satisfy all hard constraints and, in agreement with them, as large a number of soft constraints as possible. The number of satisfied soft constraints is usually a parameter for ranking alternative partial solutions.

As for SAT, many algorithms designed for CSPs can be adapted to MaxCSP. The best candidates are those based on local search and multiple restart, such as WalkSAT. Nevertheless, the literature shows that a remarkable number of specific algorithms have been produced for MaxCSP. In particular, most stochastic algorithms based on genetic algorithms and simulated annealing are aimed at solving problems that fit into the MaxCSP framework.

4.1.5 Constraint logic programming

Constraint logic programming is a form of constraint programming in which logic programming is extended to include techniques from constraint satisfaction (Van Hentenryck, 1989; Jaffar and Maher, 1994). In practice, constraint logic programming is a good framework for CSPs: logic programming provides a satisfactory environment for specifying CSPs while constraint programming algorithms provide the tools for implementing efficient CSP solvers (Apt, 2003; Dechter, 2003).

The logic programming environment is based on Horn clauses and contains DATALOG as a special case. The classical logic programming environment is represented by Prolog. In Prolog, a logic program consists of a set of Horn clauses that, in principle, can be seen as a statement of the constraints characterizing a CSP. The Prolog interpreter executes a logic program that tries to solve the CSP it encodes, i.e., it tries to find a value for each variable occurring in the clauses that satisfies all logical constraints.

The classical interpreter is based on a pure backtracking algorithm, which operates exactly as described in Section 4.1.1. In constraint logic programming the interpreter is extended with constraint propagation techniques. However, in this environment constraints on the variable domains are not explicitly given but

Backtrack algorithms

may occur in the body of a clause together with other literals. The job of discovering and handling them is left to the interpreter. Before assigning a tentative value to a variable occurring in a clause, the interpreter scans the other clauses looking for constraints on the variable's domain. Constraints encountered during this scan are kept in a set called a *constraint store*. If this set is found to be unsatisfiable, because some variable has an empty domain, the interpreter backtracks and tries to use other clauses to attain the goal. In practice the constraint set may be checked using any algorithm used for CSPs.

Constraint logic programming is still an active research area, and the environments for constraint logic programming are becoming more and more powerful and sophisticated (Apt and Wallace, 2007).

4.2 Generative models for CSPs

In order to investigate the emergence of a phase transition in CSPs, it is necessary to analyze a large number of problems, as in the case of k-SAT. However, now the problem of generating random CSPs is more complex. The problem space is characterized by a greater number of dimensions, and it becomes more difficult to explore it systematically. For this reason many authors have proposed generative models that sample restricted subspaces of the problem space in which phase transitions are detected but which are still representative of the CSPs encountered in the real world.

In the following we will consider only binary CSPs and will mostly follow Prosser's approach (Prosser, 1996). In order to generate a random ensemble of CSPs, the stochastic construction of the problem instances must follow a precise probability model. Randomly generated (binary) CSPs are characterized by the 4-tuple (n, d, p_1, p_2), where n is the number of variables, d is the size of the domains, which is assumed to be the same for all variables, p_1 is the fraction of existing edges among the $n(n-1)/2$ possible edges in the constraint graph, and p_2 is the fraction of value pairs excluded by each relation R_h; again, it is assumed that each relation prohibits the same number of pairs. The parameter p_1, the *constraint density*, may be thought of as the density of the constraint graph, whereas p_2 is the *constraint tightness* (Smith and Dyer, 1996; Prosser, 1996). We may notice that this generative model makes the simplifying assumption that all the domains in which the variables take values have the same cardinality d and that each relation R_h rules out the same number of value pairs.

Generating models

Constraint density and tightness

The 4-tuple (n, d, p_1, p_2) can be interpreted in different ways according to whether p_1 and p_2 are treated as probabilities or as parameters specifying exactly the number of edges in the constraint graph and the cardinality of the relations, respectively. More precisely, four models were introduced initially in the study

of phase transitions in CSPs (Smith and Grant, 1997; MacIntyre *et al.*, 1998; Prosser, 1996; Gent *et al.*, 2001), as follows.

Model A **Model A** In this model, both p_1 and p_2 are treated as probabilities. More specifically, given a number n of variables and a probability p_1, let the constraint graph $\boldsymbol{G}$ be an element of the Erdös and Rényi graph ensemble $\mathcal{G}_{n,p_1}$, as introduced in Definition 3.6. As described in Section 3.2, the number m of edges in $\boldsymbol{G}$ is a stochastic variable that follows a binomial distribution with mean $\overline{m} \equiv \mathbb{E}[m] = p_1 n(n-1)/2$ and variance $\mathbb{V}[m] = p_1(1-p_1)n(n-1)/2$. The relation between m and p_1 is then

$$p_1 = \frac{2\,\mathbb{E}[m]}{n(n-1)} \qquad \text{or} \qquad \mathbb{E}[m] = p_1 \frac{n(n-1)}{2}.$$

For the second control parameter, p_2, we proceed as follows: consider in turn each edge in $\boldsymbol{G}$. Let (x_i, x_j) be an edge. From the Cartesian product $\mathbf{D} \times \mathbf{D}$, of cardinality d^2, extract without replacement and with probability $1-p_2$, a number of variable assignment pairs (a_i, a_j) and use them to create a table $R_h(x_i, x_j)$. This table represents the allowed assignments (a_i, a_j) to (x_i, x_j). Then the cardinality N of each relation (edge) is a stochastic variable with mean $(1-p_2)d^2$ and variance $p_2(1-p_2)d^2$. More precisely,

$$p_2 = 1 - \frac{\mathbb{E}[N]}{d^2} \qquad \text{or} \qquad \mathbb{E}[N] = (1-p_2)d^2.$$

In model A, two instances of CSPs having the same control parameters p_1 and p_2 will differ in the number of edges in the constraint graphs, in the identity of these edges, in the number of goods, and in their identity.

Model B **Model B** In this model both p_1 and p_2 are treated as proportions (Palmer, 1985). More specifically, given a number n of variables and a number m of edges, let the constraint graph $\boldsymbol{G}$ be an element of the Erdös and Rényi graph ensemble $\mathcal{G}_{n,m}$, as introduced in Definition 3.7. In this case edges are added to $\boldsymbol{G}$ by extracting without replacement m elements from the possible set of $n(n-1)/2$ edges. The relation between m and p_1 will then be

$$p_1 = \frac{2m}{n(n-1)}.$$

For p_2, consider in turn each edge in $\boldsymbol{G}$. Let (x_i, x_j) be an edge. From the Cartesian product $\mathbf{D} \times \mathbf{D}$, of cardinality d^2, extract without replacement N variable assignment pairs (a_i, a_j) and create a table $R_h(x_i, x_j)$. These pairs represent the allowed assignments (a_i, a_j) to (x_i, x_j). Then the cardinality N of each relation is linked to p_2 via

$$p_2 = 1 - \frac{N}{d^2}.$$

In model B, two instances of CSPs having the same values of m and N will differ in the identity of the edges in the constraint graph and in the identity of the goods (see Section 4.1).

Model C In this model, p_1 is considered as a probability and p_2 as a proportion; p_1 and N are given. The constraint graph $\boldsymbol{G}$ is built up as in model A and belongs to $\mathcal{G}_{n,p_1}$. The relation between m and p_1 is Model C

$$p_1 = \frac{2\,\mathbb{E}[m]}{n(n-1)} \qquad \text{or} \qquad \mathbb{E}[m] = p_1 \frac{n(n-1)}{2}.$$

The relations corresponding to the constraints, however, are built up as in model B. Thus

$$p_2 = 1 - \frac{N}{d^2}.$$

In model C, two CSP instances having the same values of p_1 and N will differ in the number of edges in the constraint graphs, in the identity of these edges, and in the identity of the goods.

Model D In this model, p_1 is considered as a proportion and p_2 as a probability; m and p_2 are given. The constraint graph $\boldsymbol{G}$ is built up as in model B and belongs to $\mathcal{G}_{n,m}$. Then the relation between m and p_1 is Model D

$$p_1 = \frac{2m}{n(n-1)} \qquad \text{or} \qquad m = p_1 \frac{n(n-1)}{2}.$$

The constraints are built up as in model A, and so

$$p_2 = 1 - \frac{\mathbb{E}[N]}{d^2} \qquad \text{or} \qquad \mathbb{E}[N] = (1-p_2)d^2.$$

In model D, two CSP instances having the same values of m and p_2 will differ in the identity of edges in the constraint graphs and in the number and identity of the goods.

4.3 Phase transition in a CSP

Using various control parameters, several studies have been performed on the emergence of a phase transition in CSPs. In particular, Prosser (1996) systematically explored the spaces of the control parameters p_1 and p_2 experimentally, using model B. As mentioned in Chapter 3, the resulting CSP corresponds to a constraint graph belonging to $\mathcal{G}_{n,m}$ and p_1 is a parameter controlling the emergence of a phase transition in the connectivity. Thus, for low values of p_1 the resulting graph $\boldsymbol{G}$ may be disconnected. Particular cases occur when p_1 and p_2

assume extreme values. For example, if $p_1 = 1$ then $\boldsymbol{G}$ is complete. If $p_2 = 1$ then $N = 0$, i.e., no pairs of values are allowed and the problem instance is certainly unsolvable; on the contrary, $p_2 = 0$ denotes a surely solvable problem instance with no disallowed pairs.

Prosser's experiments

In the experiments Prosser used FC-CBJ-DKC-FP, a complete forward-checking algorithm with conflict-directed backjumping, extended with directed k-consistency and the fail-first heuristic (Prosser, 1996). He generated sets of problems $(20, 10, p_1, p_2)$ with $n = 20$ variables and uniform domain size $d = 10$. The values of p_1 ranged from 0.1 to 1 in steps of 0.1, whereas the values of p_2 ranged from 0.01 to 0.99 in steps of 0.01. For each pair of p_1 and p_2 values, 100 problems were generated. Globally the experiments involved $99\,000$ problems, with a ratio 1.27 of solvable and unsolvable problems.

The experiments were performed by varying the p_2 values for each p_1 value. Prosser measured the complexity of the search by the number of consistency checks; when $p_1 = 0.5$ he found that the search complexity showed a marked increase around $p_2 \approx 0.30$, reached a maximum at $p_2 \approx 0.38$, and then decreased again (see Figure 4.5).

Mushy region

In the region $0.35 < p_2 < 0.41$ there is a mixture of solvable and unsolvable problems; Smith (1994) referred to this as the *mushy region*. It is in this region that the average search effort is maximal. Unsolvable instances require, on average, a greater computational complexity at the phase transition because the whole search tree may need to be explored.

As p_2 is varied across the mushy region, the probability of finding a solution P_{sol} drops from almost 1 to almost 0 (Williams and Hogg, 1994; Smith and Dyer, 1996; Prosser, 1996). The complexity involved in finding one solution (or of proving unsolvability) shows a marked peak at $P_{sol} = 0.5$, which is called the *crossover point* (Crawford and Auton, 1996; Smith, 1994).

Prosser (1996) also performed experiments in which he let both p_1 and p_2 vary at the same time, obtaining the graphs in Figure 4.6.

The p_2 value corresponding to the crossover point, $\hat{p}_{2,cr}$, is called the *critical* value; it is conjectured to correspond to an expected number of solutions roughly equal to 1 (Williams and Hogg, 1994; Smith and Dyer, 1996; Prosser, 1996; Gent and Walsh, 1996). Its location also depends upon the structure of the constraint graph. Williams and Hogg (1994), Prosser (1996), and Smith and Dyer (1996) all derived the same estimate for the critical value of p_2:

$$\hat{p}_{2,cr} = 1 - d^{-2/p_1(n-1)} = 1 - d^{-n/m}. \tag{4.2}$$

Location of the phase transition

The estimate $\hat{p}_{2,cr}$ can be used to predict the location of the phase transition. Formula (4.2) was derived by assuming that the average number of solutions (over the set of problems) at the phase transition is 1. However, the value $\hat{p}_{2,cr}$ given by (4.2) is a good predictor only for high values of p_1 or for large values

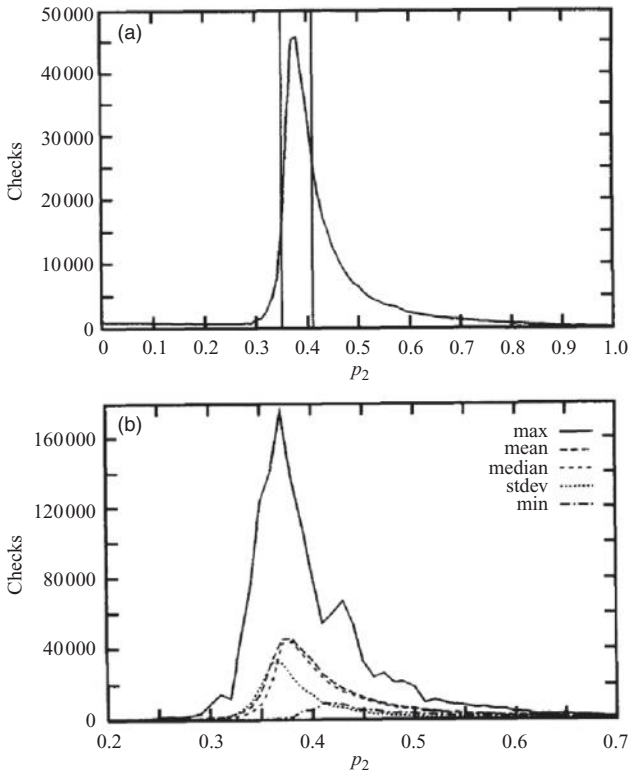


Figure 4.5 Computational search complexity *vs*. p_2 for the $(20, 10, 0.5, p_2)$ ensemble of CSP problems. (a) The mushy region. (b) The complexity is evaluated as the number of consistency checks made by the FC–CBJ–DKC–FP algorithm. For $p_2 < 0.35$ all the considered problems were solvable whereas for $p_2 > 0.41$ all were unsolvable. Reprinted from Prosser (1996) with permission.

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 $\hat{p}_{2,cr}$ and the actual location of the phase transition. In this case the mushy region is determined by a mixture of alternative constraint graphs, each corresponding to a different crossover point. An average number of solutions equal to 1 may then correspond to a large number of unsolvable problems coupled with a small number of solvable problems with many solutions. The effect is to shift the value given by (4.2) to the right of the actual phase transition location.

Model of Williams and Hogg

Other authors have used different control parameters to specify the location of the phase transition. For instance, Williams and Hogg (1994) considered an

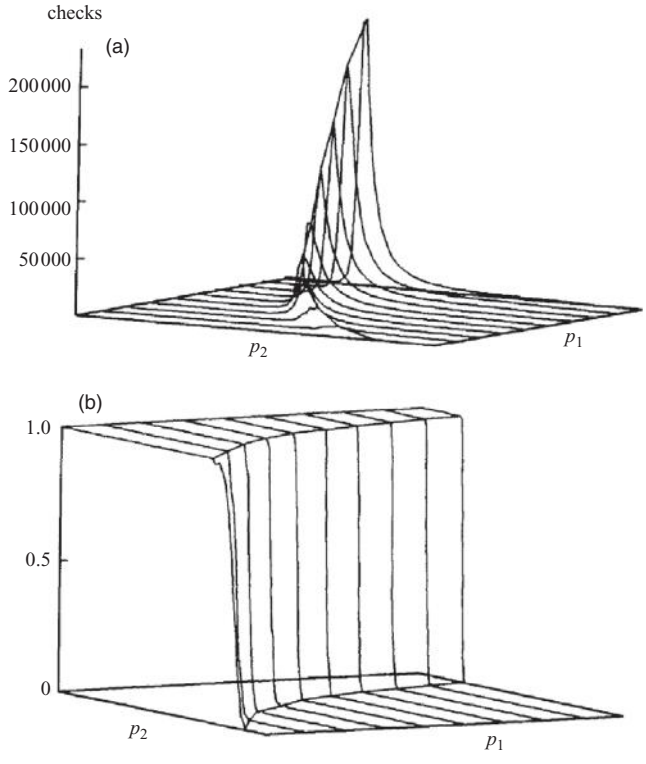


Figure 4.6 A map of (a) search effort and (b) solubility for the (20, 10, p_1, p_2) problems investigated by Prosser. Reprinted from Prosser (1996) with permission.

ensemble of problems characterized by the 4-tuple (n, d, m, N'), where n, d, and m have the same meaning as previously whereas N' is the number of nogoods per constraint, i.e., $N' = d^2 - N$. These authors suggested that the phase transition occurs when the ratio β between the total number of local nogoods (conflicts between a pair of variables) and the number of variables assumes a critical value. The parameter β can be expressed in terms of p_1 and p_2 as

$$\beta = \frac{(d^2 - N)m}{n} = \frac{1}{2}p_1(n-1)p_2d^2. \tag{4.3}$$

The critical value β_{cr} was given by Williams and Hogg (1994):

$$\beta_{cr} = -\frac{\ln d}{\ln(1 - d^{-2})}. \tag{4.4}$$

The value β_{cr} can be rewritten in terms of p_2 as follows:

$$\beta_{cr} = -p_2 d^2 \frac{\ln d}{\ln(1-p_2)}, \tag{4.5}$$

When $\beta = \beta_{cr}$, the value $\hat{p}_{2,cr}$ can be inserted into both (4.3) and (4.5), giving

$$\frac{1}{2} p_1 (n-1) \hat{p}_{2,cr} d^2 = -\hat{p}_{2,cr} d^2 \frac{\ln d}{\ln(1-\hat{p}_{2,cr})}. \tag{4.6}$$

By simplifying (4.6) and solving with respect to $\hat{p}_{2,cr}$, we obtain

$$\ln(1-\hat{p}_{2,cr}) = \frac{2}{p_1(n-1)} \ln\left(\frac{1}{d}\right). \tag{4.7}$$

and finally

$$\hat{p}_{2,cr} = d^{-2/p_1(n-1)} = 1 - d^{-n/m}. \tag{4.8}$$

Expression (4.8) is the same as (4.2).

Gent and Walsh (1996) proposed yet another parameter, κ, to quantify the constrainedness of the search. This parameter is defined in terms of the number S of search states and the average number of solutions, N_{sol}: Parameter κ

$$\kappa = 1 - \frac{\log_2 \mathbb{E}[N_{sol}]}{\log_2 S}, \tag{4.9}$$

where $\mathbb{E}[N_{sol}]$ is the expected number of solutions existing in a search space with S states. Assuming that the phase transition occurs for $\mathbb{E}[N_{sol}] = 1$, the critical value of κ is $\kappa_{cr} = 1$. For a CSP of the type considered by Gent and Walsh, formula (4.9) gives

$$\kappa = -\frac{m \log_2(1-p_2)}{n \log_2 d}. \tag{4.10}$$

Setting $\kappa_{cr} = 1$, expression (4.2) is obtained for the corresponding $\hat{p}_{2,cr}$.

4.3.1 Asymptotic behavior

In the studies reported above the number of variables was usually kept constant, whereas the structural parameters were varied. An interesting question is what happens when the CSP's size grows, in particular when the number of variables $n \to \infty$.

In some insightful work, Achlioptas *et al.* (1997, 2001a) investigated the CSP ensembles generated with model B by assuming that the number of variables, n, increases. The main result is that when $n \to \infty$ the probability that a random

Asymptotic behavior for $n \to \infty$

instance generated by Model B is unsolvable tends to 1, provided that $p_2 \geqslant 1/d$. Thus, the authors question the claim that a "true" phase transition exists for such ensembles (except for a small region in the parameter space where $p_2 < 1/d$). The condition $p_2 < 1/d$ is equivalent to

$$1 - \frac{N}{d^2} < \frac{1}{d} \qquad \Longleftrightarrow \qquad N > d(d-1).$$

More specifically, Achlioptas *et al.* (1997) set $p_1 = c/n$, where c is a sufficiently large constant. Using the expression $p_1 = 2m/n(n-1)$ for model B one obtains

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

and thus

$$np_1 = \frac{2m}{n-1} = c = \text{constant}.$$

As a consequence the number of constraints, m, must scale linearly with n.

The asymptotic behavior of CSPs for increasing n was investigated also by Smith (2001), who proposed a modification of model D in order to obtain a guaranteed phase transition in CSPs, for a specified range of p_2 values. This was achieved by letting some parameters defining the instances vary with n, as described in the next subsection.

4.3.2 New models

In order to overcome the problem discussed in the previous subsection, new models for generating CSP ensembles, which are guaranteed to exhibit a phase transition even asymptotically, have been proposed. One was defined by Achlioptas *et al.* (1997); the authors called it model E.

Model E

Model E This model is specified by four parameters n, m, d, k, where n is the number of variables, m the number of constraints, d the cardinality of the variables' domains, and k the arity of the constraints (we will consider the case $k = 2$). The CSP instances are generated by selecting uniformly, at random, and with repetition a total number of nogoods.

The generation of nogoods in model E is similar to that proposed by Williams and Hogg (1994) except that repetition is allowed in model E. According to Achlioptas *et al.* (1997), model E is interesting when the number of constraints is $m = \Theta(n)$. However, a disadvantage of this model is that a complete constraint graph is easily generated, in which constraints have only a few forbidden pairs of values. This makes it unrepresentative of the CSPs that arise in real-world problems.

Denoting by r the ratio m/n of the number of constraints over the number of variables, and by $P_k(n, r)$ the probability that a random instance generated by model E be solvable, the following properties can be proved.

Exact phase transition

Theorem 4.1 {*Achlioptas* et al., *1997*} *For any k there exists a function $r_k(n)$ such that, for any $\epsilon > 0$,*

$$\lim_{n\to\infty} P_k(n, r_k(n) - \epsilon) = 1 \qquad \textit{and} \qquad \lim_{n\to\infty} P_k(n, r_k(n) + \epsilon) = 0.$$

Then, for all $r \geqslant 0$ and $\epsilon > 0$,

$$\textit{if} \qquad \liminf_{n\to\infty} P_k(n, r) \geqslant \epsilon \qquad \textit{then} \qquad \lim_{n\to\infty} P_k(n, r) = 1.$$

The above theorem can be used to prove that ensembles of CSP instances generated according to model E indeed exhibit a phase transition. However, Gent *et al.* (2001) showed that things are actually more complex than that, as there are regions where model B does not suffer from the flaws pointed out by Achlioptas *et al.* (1997) whereas model E does. In the same paper, the authors noticed also that in real-world problems it is not unusual to observe structures in a constraint graph that are very rare in random graphs. These structures may strongly affect the performances of solving algorithms.

Modified model D

As mentioned in the previous subsection, Smith (2001) proposed a modification of model D in which both the number of constraints m and the domain size d slowly increase with n. More specifically, let us suppose that we have generated an ensemble of instances having parameter sets $(n_0, m_0, p_1^{(0)}, p_2^{(0)})$. If $n \to \infty$, on the one hand the behavior of $d(n)$ is determined by the expression

$$d = a \log n + c,$$

where a is a constant greater than 1, the log is to the base 2, and c is a constant depending on n_0, m_0, and a. On the other hand, $m(n)$ is determined by the following equation:

$$\left\{1 - \left[1 - \left(1 - p_2^{(0)}\right)^d\right]^m\right\}^{n-1} = \frac{1}{2}. \tag{4.11}$$

From (4.11) the function $m(n)$ can be derived:

$$m(n) = \frac{\log\left(1 - 2^{-1/(n-1)}\right)}{\log\left[1 - \left(1 - p_2^{(0)}\right)^{d(n)}\right]}.$$

With this choice of $d(n)$ and $m(n)$, all instances for which $p_2 \leqslant p_2^{(0)}$ are satisfiable with probability at least 0.5. Hence, $p_2^{(0)}$ is a lower bound for the transition

point. In order to guarantee that a phase transition occurs for all values of n, the upper bound of $p_{2,cr}$ must converge to a value less than 1. This last condition was verified experimentally by Smith (2001), who also proved that

$$p_{2,cr} < 1 - \left(1 - p_2^{(0)}\right)^2 .$$

The above results, obtained by modifying model D, cannot be transferred directly to the other standard models, A, B, and C.

Starting from the analysis of Achlioptas *et al*. (1997), Xu and Li (2000, 2006) proposed a new generative model, model RB, which is guaranteed to show a phase transition even in the asymptotic case of large n. The model has two control parameters, r and p; they have critical values r_{cr} and p_{cr}, respectively. For any fixed value $r < r_{cr}$ or $p < p_{cr}$, a random CSP instance, generated by model RB, is satisfiable with probability almost 1, when $n \to \infty$. Conversely, a random CSP instance is unsatisfiable with probability almost 1 when $r > r_{cr}$ or $p > p_{cr}$. The critical values r_{cr} and p_{cr} can be computed exactly. Model RB assumes that all constraints have arity $k \geqslant 2$. The main departure from previous models is that the cardinality of the variables' domains is no longer constant but depends on n: $d = n^{\alpha}$. The generation of the CSP instances is achieved in two steps, as described in the following.

Model RB

Model RB (Xu and Li, 2000) *Step 1*: Select *with* replacement $m = rn \ln n$ random constraints, each involving a set of k variables extracted *without* replacement from the set $\mathbf{X}$. *Step 2*: For each constraint, select uniformly, *without* replacement, a number $N = (1-p)d^k$ of compatible tuples (goods) of values.

An instance of model RB is denoted by $\mathrm{RB}(k, n, \alpha, r, p)$. The parameter $r > 0$ determines the number of constraints $m = rn \ln n$, the parameter p $(0 \leqslant p \leqslant 1)$ determines the number $N = (1-p)d^k$ of allowed tuples for each relation, $k \geqslant 2$ is the arity of each relation, $n \geqslant 2$ is the number of variables, and $\alpha > 0$ determines the domain size $d = n^{\alpha}$ of each variable.

Regarding model RB, the main results of the papers of Xu and Li (Xu and Li, 2000, 2006) is contained in the theorems that follow.

Exact location of the phase transition

Theorem 4.2 $\{$*Xu and Li, 2000*$\}$ *Let* $r_{cr} = -\alpha/\ln(1-p)$. *If* $\alpha > 1/k$ *and* $0 < p < 1$ *are two constants, and* k *and* p *satisfy the inequality* $k \geqslant 1/(1-p)$, *then*

$$\lim_{n\to\infty} \mathrm{P}(SAT) = 1, \qquad r < r_{cr}, \tag{4.12}$$

$$\lim_{n\to\infty} \mathrm{P}(SAT) = 0, \qquad r > r_{cr}. \tag{4.13}$$

Theorem 4.3 *{Xu and Li, 2000}* *Let $p_{cr} = 1 - e^{-\alpha/r}$. If $\alpha > 1/k$ and $r > 0$ are two constants, and k, α, and r satisfy the inequality $ke^{-\alpha/r} \geqslant 1$, then*

$$\lim_{n\to\infty} \mathtt{P}(SAT) = 1, \qquad p < p_{cr}, \tag{4.14}$$

$$\lim_{n\to\infty} \mathtt{P}(SAT) = 0, \qquad p > p_{cr}. \tag{4.15}$$

For the problem instances generated by model RB it is possible to compute an average number of solutions:

$$\mathbb{E}[sol] = d^n(1-p)^{rn\ln n} = n^{\alpha n}(1-p)^{rn\ln n}.$$

With model RB it is possible to generate provably hard instances in the proximity of the phase transition location for all values of n (Xu and Li, 2006).

4.4 Comments

Solving CSP is central to the theme of the book; relational learning heavily relies on it, as will be described in Chapters 9 and 10. Even though any finite CSP can be translated into an equivalent SAT problem, as will be shown in Section 8.4, handling CSP instances directly appears more complex than handling SAT instances; in the first place CSPs have more degrees of freedom (two relevant control parameters, p_1 and p_2, instead of just one, α, as in SAT). Moreover, several generative models with different characteristics are available.

An interesting point, not yet fully analyzed,[1] is the structure of the constraint graph. Basically, in the vast majority of work this structure corresponds, with small variations, to the Erdös and Rényi ensembles of random graphs (see Section 3.2), but in principle the differing underlying structures can deeply affect solvability. As in the case of SAT, if we consider CSP instances that are not random then the consequences may be quite different from the theoretical predictions, both in terms of solvability and in terms of computational complexity.

Moreover, we have to notice that statistical physics methods have been applied directly to CSPs much more rarely than to SAT problems, and mainly with the aim of counting existing models. A paper that makes use of these methods is Gent *et al.* (1995). The authors established a (weak) link with a physical system by associating variables with multi-valued "spins" interacting through the CSP's relations and associating the parameter κ (see (4.10)) with a kind of "temperature". They used a heuristic method called *finite-size scaling* to model the phase transition in CSPs. This method predicts that, at the phase transition, problems

[1] With some exceptions; see, for instance (Walsh, 1999).

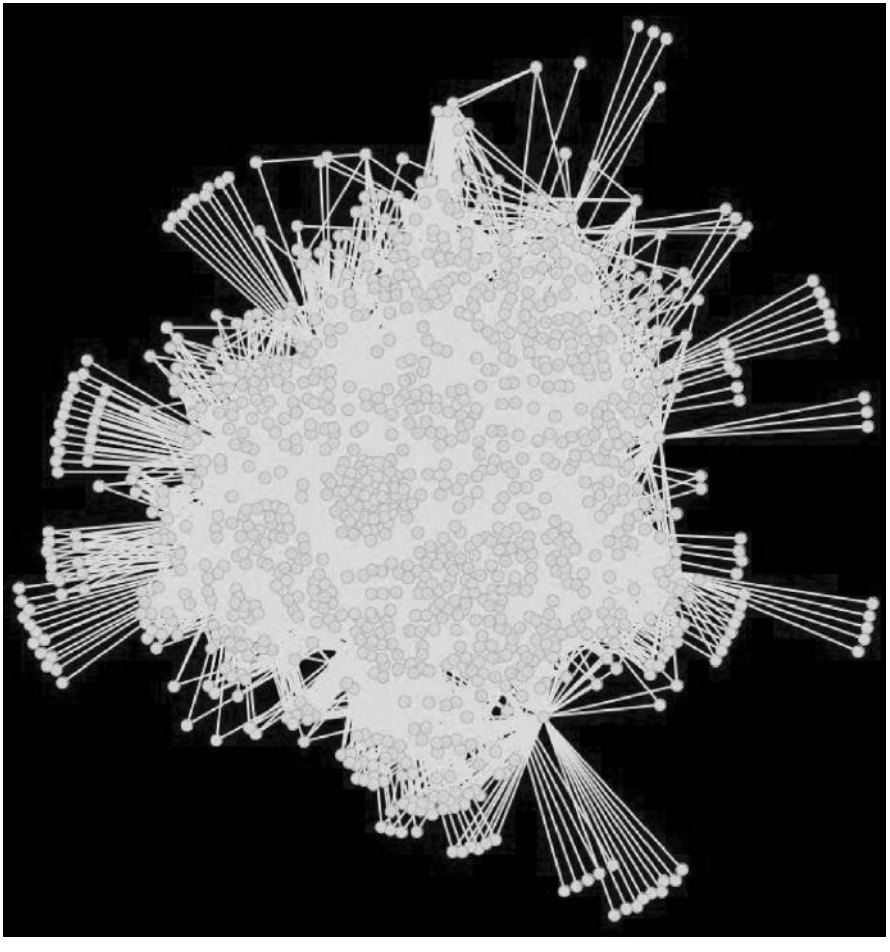


Figure 4.7 Example of a graph associated with a CSP problem having $m = 12$ literals and $L = 15$ constants.

of different sizes cannot be distinguished, except for a change of scale. In other words, we have

$$P_{sol} = f\left((\tau - \tau_{cr})\, N^{(1/\nu)}\right), \tag{4.16}$$

where f is some function, τ is the order parameter under consideration, and N represents the size of the system.

A closer connection between a CSP and a statistical physics system might be established by associating goods with the particles in a many-body system.

Each good corresponds to a node in a graph; the different goods (a_i, a_j), satisfying a predicate $\varphi(x_i, x_j)$ are independent and, hence, not connected; two goods (a_i, a_j) and (b_h, b_k) satisfying predicates $\varphi_1(x_i, x_j)$ and $\varphi_2(x_h, x_k)$, respectively, are connected in the graph if the conjunction $\varphi_1(a_i, a_j) \wedge \varphi_2(b_h, b_k)$ is true. An example of such a graph is shown in Figure 4.7. Finding a solution to the CSP corresponds to finding a subgraph in this graph (see Chapter 14 for more details). The graph obtained may be quite complex and can be analyzed using statistical physics methods, as described in Chapter 12.

5

Machine learning

Contents

In this chapter we introduce the main topic of this book, namely machine learning. In order to make the book self-contained, a brief introduction to the subject is presented.

Learning

The word *learning* is generally associated with a category of performance, as in *learning to play chess* or *learning to program computers*, or with a corpus of knowledge, as in *learning geometry*. It also often comes in association with some mechanism, as in *learning by doing*, *learning by explanation*, *learning by analogy*, *learning by trial and error*, and so on.

Learning appears under seemingly very different guises: from *habituation* (the reduction in sensibility or attention with the repetition of a situation or in problem solving), *imitation*, *rote learning*, *discrimination* (the ability to distinguish one type of input from others), *categorization* (constructing categories from observations of the environment), to learning *whole bodies of knowledge* or even *discovering theories* about the world.

Ubiquity of learning

Since learning (throughout evolution or through cultural and individual acquisition) is at the source of all behavioural and cognitive capabilities in natural systems, as well as in many artificial ones, it is no surprise that it displays such a wide variety of appearances and mechanisms. There are, however, some common aspects and ingredients that underlie many of these diverse manifestations

of learning. One can list at least three:

- percepts
- decision and actions
- measure of performance

An agent learns when it interacts with the world, using percepts to make decisions and take actions, and then measures its performance with the aim of improving its future decision-making process.

Rote learning is the limiting case, when learning solely involves the storage of past experience in order to determine future decisions or actions. Actually, even rote learning is not possible *per se* since it implies by necessity some form of selection in the storage of the "raw percepts". This in turn means that several percepts might be stored as the same "experience" even though they are actually different. *Generalization*, albeit in an arguably limited form in rote learning, is thus an intrinsic part of learning. This is even more so when one is considering learning to discriminate from examples, categorizing, or constructing theories about the world.

Generalization

One of the simplest and purest form of learning is *concept learning*. The word "concept" may have several meanings and definitions. In the first place there are two approaches to defining a concept, the extensional approach and the intentional approach. In the extensional approach a concept is simply a set of objects (instances) that share some properties. In the intensional approach there are at least three different views. In the classical view, which goes back to Aristotle, a concept is a name corresponding to a set of necessary and sufficient conditions. The instances are not given explicitly but they are all those objects that satisfy the conditions. This definition, well suited for mathematical concepts, is inadequate for everyday-life concepts, as Wittgenstein pointed out with the simple example of the concept of a "lemon", for which defining criteria cannot be formulated (Wittgenstein, 1954). Thus weaker definitions have been introduced. The *"heuristic" view of a concept keeps only sufficient conditions*, whereas the "exemplar" view (Rosch, 1973; Murphy, 2002) considers a concept as a prototype, i.e., a real or virtual example characterized by the "typical" features occurring in the instances. In machine learning the heuristic view dominates the field.

Concepts

5.1 Concept learning

Interesting experiments on human and animal classification in regard to toxic substances were reported by Fabre-Thorpe *et al*. (2001).

In many learning situations a central task consists in acquiring the ability to distinguish between one class of patterns (a "concept") and the rest of the world. Evolution has thus endowed animals with the ability to quickly distinguish

predators from other living beings and to distinguish edible substances from potentially toxic substances. In most cases this discrimination capability is acquired through exposure to experiences felt either as positive (*e.g.*, good food) or as negative (*e.g.*, bringing sickness). This type of learning situation, where each supposedly isolated input is associated with a response, is called *supervised learning* because it corresponds to teacher-assisted environments, in which a teacher helps the learner by tagging each output. The couple (*input, tagged output*) is accordingly called a *training instance*.

Supervised learning

Training examples

When the responses, or tagged outputs, can only take one of two values (positive or negative), the learning task is described as *concept learning*. Training instances associated with a positive response are called *positive examples*, or *examples* for short, while instances associated with a negative response are called *negative examples* or *counter-examples*.

More formally, concept learning involves the task of inducing a Boolean function defined over an input space, also called an *example space* and denoted by $\mathcal{X}$, onto a set $\mathcal{Y} = \{-1, 1\}$ from a limited training set, denoted by $\mathcal{S}_L$, comprising m learning examples $(\vec{\mathbf{x}}_i, y_i)$ $(1 \leq i \leq m)$. Although the term "set" is conventional, these training examples may be either different examples or repeated examples. Furthermore, they can be accessed and considered sequentially by the learner – in an *online setting* – or dealt with in a single sweep, as in a *batch setting*. One crucial difference lies in the fact that, in the former setting, at any one time the learner maintains a model of the environment and has to use this model, instead of a memory of past events, to make decisions and learn further.

5.1.1 A formal view of concept learning

Most studies in inductive learning assume that the patterns that are encountered, either by the learner or the classifier, are generated according to a random process, as follows. Each pattern or object belongs to class -1 or class 1 and is summarized by a finite number, say n, of measurements (generally real-valued), called *features*. Thus the description of a pattern is given by a *feature vector* $\vec{\mathbf{x}} \in \mathcal{X}$ (where, most often, $\mathcal{X} = \mathbb{R}^n$). The uncertainty about the class to which an object belongs is modeled using *a priori* probabilities P_{-1} and P_1 for the two classes (such that $P_{-1} + P_1 = 1$). To express the relationship between the class of an object and the feature vector (including the uncertainty and noise in the measurement process), it is assumed that an object in the class $y \in \{-1, 1\}$ engenders a random feature vector according to a class-conditional distribution function $P_{\mathcal{X}|\mathcal{Y}}$ (see Figure 5.1).

In this setting the learning agent comes across random feature vectors $\vec{\mathbf{x}}$ (called "observables"), which are generated according to the following two-stage

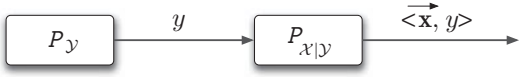


Figure 5.1 The two-stage generation of learning examples.

process. First, a random class $y \in \{-1, 1\}$ is selected using the *a priori* probabilities $P_{\mathcal{Y}}$; then the observed feature vector $\vec{\mathbf{x}}$ is generated according to the class-conditional distribution $P_{\mathcal{X}|\mathcal{Y}}$. The distribution over labeled patterns is thus given by $P_{\mathcal{X}\mathcal{Y}}(\vec{\mathbf{x}}, y) = P_{\mathcal{Y}}(y)\, P_{\mathcal{X}|\mathcal{Y}}(\vec{\mathbf{x}}, y) = P_{\mathcal{X}}(\vec{\mathbf{x}})\, P_{\mathcal{Y}|\mathcal{X}}(\vec{\mathbf{x}}, y)$.

When acting as a *classifier*, the agent is facing the following problem: given a realization of the measured feature vector $\vec{\mathbf{x}}$, to decide whether the unknown object engendering $\vec{\mathbf{x}}$ belongs to class -1 or class 1. In this setting, a *classifier* or *decision rule* is simply a mapping $h : \mathcal{X} \to \mathcal{Y}$ that determines the class $h(\vec{\mathbf{x}})$ to which an observed feature vector $\vec{\mathbf{x}}$ should be assigned. In the context of machine learning this map is called a *hypothesis*, hence the notation h. We will denote by $\mathcal{H}$ the space of possible hypotheses.

In statistics a *hypothesis* is known as a *model*.

The performance of a classifier can be defined as the *probability of error*, given by

$$\epsilon(h) = P_{\mathcal{X}\mathcal{Y}}\{h(\vec{\mathbf{x}}) \neq y\}. \tag{5.1}$$

In general different costs can be assigned to different types of errors by specifying a *loss function* ℓ, defined as

In medicine it is much more costly to miss a tumor diagnosis than to make a more expensive analysis only to find out that it was a false alarm.

$$\ell(h(\vec{\mathbf{x}}), y) : \mathcal{X} \times \mathcal{Y} \;\; \to \;\; \mathbb{R}^+. \tag{5.2}$$

The performance of a classifier is defined as a *risk*, which is an expectation over the possible events:

Risk.

$$R(h) = \mathbb{E}[\ell(h(\vec{\mathbf{x}}), y)] = \int_{\vec{\mathbf{x}} \in \mathcal{X}, y \in \mathcal{Y}} \ell(h(\vec{\mathbf{x}}), y) P_{\mathcal{X}\mathcal{Y}} d(\vec{\mathbf{x}}, y). \tag{5.3}$$

If the *a priori* probability $P_{\mathcal{Y}}$ and conditional probability $P_{\mathcal{Y}|\mathcal{X}}$ are known then the optimal decision rule, in the sense that it gives the minimum probability of error (or minimum risk), is *Bayes' decision rule*,[1] denoted h^* and defined by

Bayes' decision rule

$$h^*(\vec{\mathbf{x}}) = \underset{y \in \{-1,1\}}{\mathrm{ArgMin}} \big(\ell(y, 1-y)\, P_{\mathcal{Y}\mathcal{X}}(\vec{\mathbf{x}}, y)\big). \tag{5.4}$$

In many situations, however, the distribution $P_{\mathcal{X}\mathcal{Y}}$ is unknown or only partially known. It is then generally assumed that, in addition to the observed

[1] The Bayes error is zero when there are no two examples, belonging to different classes, that have identical descriptions. When such two examples do exist, it is clearly impossible to discriminate between them.

vector $\vec{\mathbf{x}}$, one is given a training set $\mathcal{S}_L = \{(\vec{\mathbf{x}}_1, y_1), \ldots, (\vec{\mathbf{x}}_m, y_m)\} \in (\mathcal{X} \times \mathcal{Y})^m$ that is chosen according to the unknown probability distribution $P_{\mathcal{X}\mathcal{Y}}$. The basic assumption underlying learning is that all the data (both observed and unseen) are generated by the same process, which is formalized by saying that the data is sampled independently from the same (fixed but unknown) probability distribution (*i.i.d. sampling*). The *i.i.d.* assumption expresses the relationship needed for the induction task of inferring rules for future unseen data from the data at hand.

The **learning problem** can be then formulated as follows. *Given a training set consisting of labeled objects, assumed to be drawn i.i.d. from the unknown distribution $P_{\mathcal{X}\mathcal{Y}}$, find a function h that assigns labels to objects such that if new objects are given then this function will label them correctly.*

5.1.2 Concept learning in three questions

Short of attaining a perfect identification of the target dependency between the feature vectors $\vec{\mathbf{x}}$ and their labels, the performance of a classifier or hypothesis is measured by the risk $R(h)$ (see (5.3)). A large part of the theory in machine learning focuses on finding conditions for constructing good classifiers h whose risk is as close to $R^* = R(h^*)$ as possible.

Thus the questions that have to be answered for learning to take place are the following.

1. How should one *choose* an appropriate hypothesis space?
2. How should one *evaluate functions* in the hypothesis space?
3. How should one *explore the space* in order to find a (sub)optimal hypothesis?

The first two questions are intimately related. Let us first tackle the second question: *on the basis of the information available after the performance of a training set has been observed, which hypothesis (or hypotheses) should be preferred?*

Empirical risk

A natural and simple approach is to consider the class $\mathcal{H}$ of hypotheses $h : \mathcal{X} \to \mathcal{Y}$ and to estimate the performance of each hypothesis on the basis of its empirical performance measured on the learning set. The most obvious choice to estimate the risk associated with a hypothesis is to measure its *empirical risk* on the learning set $\mathcal{S}_L$:

$$R_m(h) \;=\; \frac{1}{m} \sum_{i=1}^{m} \ell(h(\vec{\mathbf{x}}_i), y_i), \tag{5.5}$$

which, in the case of binary classification with a $\{0, 1\}$ loss function, gives

$$R_m(h) \;=\; \frac{1}{m}\sum_{i=1}^{m}\mathbb{I}_{(h(\vec{\mathbf{x}}_i)\neq y_i)}, \tag{5.6}$$

where one counts the number of prediction errors on the training set. In (5.6), $\mathbb{I}_\alpha$ denotes the *indicator function* of the condition α:

$$\mathbb{I}_\alpha = \begin{cases} 1 & \text{if } \alpha \text{ is } \textit{true}, \\ 0 & \text{if } \alpha \text{ is } \textit{false}. \end{cases}$$

In this framework it is natural to select the hypothesis with the minimal empirical risk as the most promising one to classify unseen events. This choice criterion is called the *empirical risk minimization* principle and stipulates the choice of a best candidate hypothesis as

$$\hat{h}^* \;=\; \underset{h\in\mathcal{H}}{\text{ArgMin}}\, R_m(h). \tag{5.7}$$

This would seem to answer the question about the so-called *inductive criterion*, i.e., how to evaluate the candidate hypotheses with respect to the learning set at hand. However, the statistical theory of learning, developed over three decades by Vapnik and co-workers (Vapnik, 1999) and flourishing nowadays, has shown that it is crucial that the hypothesis space from which candidate hypotheses are drawn should be limited in terms of its expressive power. Otherwise, in the limit one can perform rote learning, and not incur any empirical risk, but obviously without real learning as such. To adapt the classifier to the training data too closely generates the phenomenon of *overfitting*, i.e., the classifier shows good performances on the training set but behaves poorly on future, unseen, data.

The studies of Vapnik and co-workers thus answer both the first and second questions, about the choice of an appropriate class of hypotheses and about the right way to evaluate the hypotheses for inductive purposes, which must take the hypothesis space into account. Namely, one should concentrate not only on finding hypotheses that minimize the empirical risk irrespective of the hypothesis space but also take into account its *capacity* or expressive power. In fact, the less diverse is $\mathcal{H}$, the tighter the link between the measured empirical risk $R_m(h)$ and the expected risk $R(h)$ for a given sample size m.

Capacity of the hypothesis space

Modern inductive techniques automatically search for the best trade-off, given the training data, following either one of two approaches:

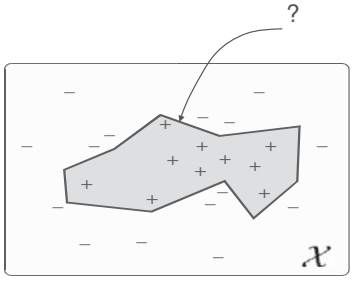


Figure 5.2 From a training set with examples (labeled $+$) and counter-examples (labeled $-$), the learner tries to find a partition of $\mathcal{X}$ that discriminates between the patterns $\vec{\mathbf{x}}$ (shaded region) that belong to the target concept and those that do not belong to it.

- *Model selection*,[2] whereby induction is considered as an alternating two-step process, in which a hypothesis space $\mathcal{H}_i$ is chosen and then a best hypothesis $\hat{h}_i^* \in \mathcal{H}_i$ is found, this procedure being repeated until a hypothesis space with lowest total error (estimation plus approximation) is found.
- *Regularization*, where one looks for a hypothesis directly minimizing a combination of the empirical error and a term depending on the capacity of $\mathcal{H}$ (or sometimes depending on the complexity of h).

We now turn to the third question: how to search effectively for a good hypothesis in the hypothesis space.

5.1.3 Searching the hypothesis space

Recall that concept learning consists of finding, within an example space $\mathcal{X}$, a region containing all and only the instances of the concept to be learned, using information provided by the learning set $\mathcal{S}_L = \{(\vec{\mathbf{x}}_1, y_1), (\vec{\mathbf{x}}_2, y_2), \ldots, (\vec{\mathbf{x}}_m, y_m)\}$. The concept to be learned is called the *target concept*. In the ideal case the examples are not corrupted by noise and the description language is rich enough that no example and its counter-example share the same description. In this case the desired region must include all examples (those with label $y = 1$) and must not include any counter-examples (those with label $y = -1$). One says that the region must *cover* the examples and must exclude the counter-examples (see Figure 5.2).

[2] In the recent proposal called *ensemble learning*, hypotheses are not selected but combined. However, such a method is not considered in this book.

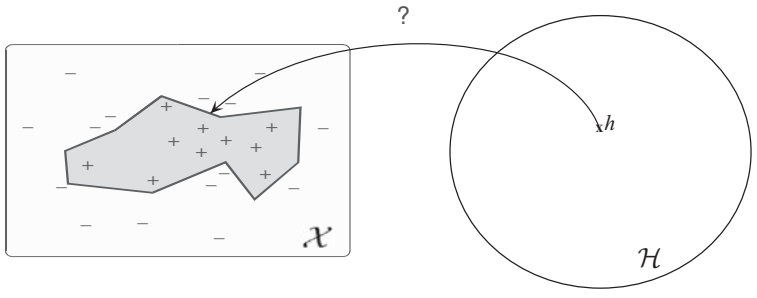


Figure 5.3 Using a hypothesis space $\mathcal{H}$. Each element h, or hypothesis, of $\mathcal{H}$ is associated with a partition of $\mathcal{X}$.

Of course, except in special cases, one is not interested in learning a concept defined in *extension*, that is by enumerating (if indeed this is possible) all its elements. One rather looks for some description of the concept in a representation language $\mathcal{L}_{\mathcal{H}}$ that defines a hypothesis space $\mathcal{H}$.

In this case, concept learning becomes the search for an expression in $\mathcal{L}_{\mathcal{H}}$, i.e., for a hypothesis $h \in \mathcal{H}$ that describes a partition in the example space $\mathcal{X}$ (see Figure 5.3) that best fits the training set. In the case of a $\{0, 1\}$ loss function, concept learning corresponds to searching in $\mathcal{H}$ for a hypothesis (and hence a partition of $\mathcal{X}$) that, as far as possible, "covers" all positive training examples and excludes all negative training examples, therefore incurring minimal empirical risk. A hypothesis associated with a subset of $\mathcal{X}$ that includes every positive example is said to be *complete*, while it is said to be *correct* if it excludes every negative example.

When a hypothesis space is available, the search for a partition in $\mathcal{X}$ is performed through an exploration of $\mathcal{H}$. Indeed, by modifying the expression of a candidate hypothesis, one obtains new hypotheses associated with new partitions of $\mathcal{X}$. In this way, learning becomes the art of searching the hypothesis space in order to identify a hypothesis that has a minimal, or near minimal, empirical risk. Using a hypothesis space has several advantages.

1. First, concepts are handled *intensionally* rather than *extensionally* (see the start of Section 6.1), which is more practical, and more helpful, in interpreting the hypothesis if the language is well chosen.

2. Second, since it is usually the case that not every partition of $\mathcal{X}$ is expressible in the representation language $\mathcal{L}_{\mathcal{H}}$, induction comes naturally. Indeed, one can no longer be satisfied with rote learning of the positive instances; one must look for the hypothesis associated with the partition closest to the available data. In a sense the more constrained the hypothesis language,

the larger the necessary inductive leap. Of course, if the hypothesis space is not well attuned to the target regularities then induction cannot come up with satisfactory hypotheses.

3. Finally, the hypothesis space can offer structures that may (or may not) be of help in conducting a systematic and efficient exploration of $\mathcal{H}$.

To look into the third point in more detail, let us suppose that, at time t, the learner is not satisfied with its current candidate hypothesis h_t; how, then, should it choose another hypothesis? It is at this point that the existing structures on $\mathcal{H}$ can play a determining role with regard to the efficiency of the learning process. The richer and the more attuned to induction are these structures, the easier becomes an efficient exploration of $\mathcal{H}$. Let us examine three possible cases, each corresponding to an increasing degree of structure in $\mathcal{H}$.

- The hypothesis space is *not endowed with any* a priori *topology or metric*. In this case only a random exploration is possible. There is nothing to guide the search. This is the worst-case scenario.

- The hypothesis space is *endowed with a neighborhood relationship*. It is then possible to explore $\mathcal{H}$ using gradient-like optimization techniques. The learner explores the neighborhood of the current hypothesis (either by something akin to differentiation, if $\mathcal{H}$ allows for this, or by the enumeration of neighbor hypotheses) and selects the direction of greatest gradient of a given criterion. This technique is very popular since it can make use of general optimization procedures. Most hypothesis spaces lend themselves to the definition of a distance and therefore of a neighborhood. However, one fundamental problem is to identify a relevant neighborhood relationship. A badly informed choice can mean that the learner goes away from the best area in the hypothesis space. Furthermore, this is still a weak structure, which, except in special circumstances (the differentiability, convexity, and other helpful properties of $\mathcal{H}$) does not yield to a fast and thorough exploration of the hypothesis space.

- In some cases, it is possible to *endow the hypothesis space with a stronger structure*, making it possible to organize its exploration efficiently. This is, in particular, the case where there are partial orderings of the elements of $\mathcal{H}$ induced by a *generality* relationship between hypotheses. In this case it becomes possible, for example, to modify an erroneous hypothesis either by specializing it just enough that it no longer covers the offending negative examples or, conversely, by generalizing it just enough that it covers the positive examples so far excluded. This type of partial ordering,

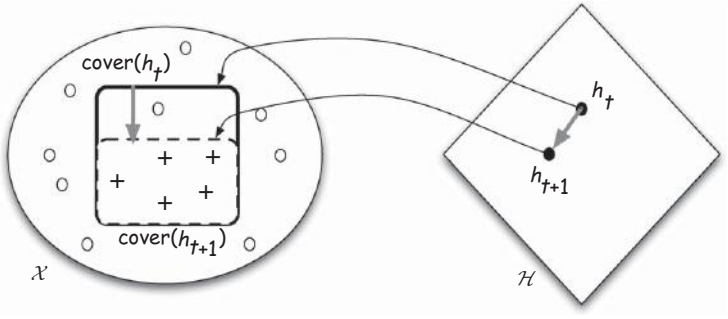


Figure 5.4 The inclusion relation in $\mathcal{X}$ induces a generality relation in $\mathcal{H}$. Here $h_{t+1} \preceq h_t$. The circles denote negative examples and the crosses denote positive examples.

induced by generality relationships, does not come naturally with every hypothesis space. For instance, if one uses a hypothesis space induced by the possible weight values of a neural network then there is no known way to decide easily whether one hypothesis is more general than another. However, some hypothesis languages, noticeably those based on logical representations, lend themselves to such partial ordering. In this case the exploration of the hypothesis space for inductive purpose is greatly expedited.

5.1.4 Hypothesis space with generality relationships

Hypothesis h_1 is said to be *more general* than hypothesis h_2 (notated $h_1 \succcurlyeq h_2$) if and only if the associated subset of h_1 in $\mathcal{X}$ (called its *coverage*) includes the subset associated with h_2. More formally, we have the following definitions.

Definition 5.1 {***Coverage of a hypothesis***} The coverage of a hypothesis $h \in \mathcal{H}$, with respect to an example space $\mathcal{X}$, is denoted by $cover(h)$, and is defined as the subset of $\mathcal{X}$ described by h. Hypothesis h is said to *cover* the elements of $cover(h)$.

Definition 5.2 {***Generality relation in*** $\mathcal{H}$} A hypothesis h_1 is *more general* (or *less specific*) than a hypothesis h_2, notated as $h_1 \succcurlyeq h_2$, if and only if $cover(h_2) \subseteq cover(h_1)$.

The relations $\preccurlyeq$ and $\succcurlyeq$ on $\mathcal{H}$ are illustrated in Figures 5.4 and 5.5.

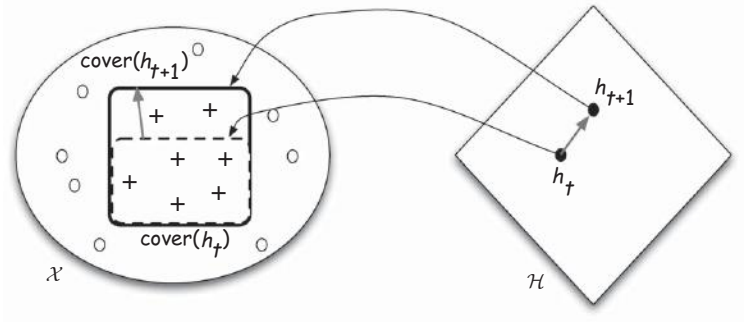


Figure 5.5 The inclusion relation in $\mathcal{X}$ induces a generality relation in $\mathcal{H}$. Here $h_{t+1} \succeq h_t$. The circles denote negative examples and the crosses denote positive examples.

The inclusion relation defined over $\mathcal{X}$ induces a generality relation over $\mathcal{H}$, which is a partial order relation. The partial order relation induces a *lattice structure* over $\mathcal{H}$. This means that, for any pair of hypotheses h_i and h_j, there exists at least one hypothesis which (a) is more general than both h_i and h_j and (b) cannot be made more specific without losing this property. The set of such hypotheses is called the set of *maximally specific generalizations* of h_i and h_j, denoted $msg(h_i, h_j)$, or the set of *least general generalizations*, denoted $lgg(h_i, h_j)$. Likewise, there exists a non-empty set of hypotheses that are more specific than h_i and h_j and that cannot be generalized without losing this property. This set is called the set of *maximally general specializations* of h_i and h_j and is denoted $mgs(h_i, h_j)$.

It is easy to extend these definitions to the case of sets of hypotheses that are larger than pairs, and we then obtain the sets $lgg(h_i, h_j, h_k, \ldots)$ and $mgs(h_i, h_j, h_k, \ldots)$.

Finally, we assume[3] that there exists in $\mathcal{H}$ a hypothesis that is more general than all others (called the *maximal* element), denoted $\top$, and a hypothesis that is more specific than all others (called the *minimal* element), denoted $\bot$ (see Figure 5.6).

However, one should be aware that the regularities existing in $\mathcal{X}$ do not, in general, translate completely to the hypothesis space $\mathcal{H}$ defined by the language $\mathcal{L}_{\mathcal{H}}$. For instance, the *least general generalization (lgg)* and the *most general specialization (mgs)*, are not, in general, uniquely defined.

[3] This assumption is valid in general.

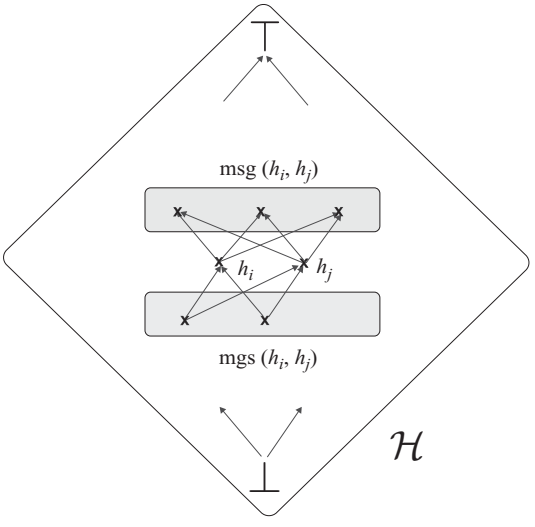


Figure 5.6 A schematic and partial view of the generalization lattice induced on $\mathcal{H}$ by the inclusion relation in $\mathcal{X}$. Each arrow corresponds to a generality relation $\preccurlyeq$ between a pair of hypotheses (crosses).

EXAMPLE

Consider the following two clauses:[4]

```
c1:  p(X,Y)  ←  q(a,f(X)),s(t(c),Y).
c2:  p(b,Z)  ←  t(Z),q(Z,f(b)),s(t(c),Z).
```

Lower-case letters stand for constants and capital letters stand for variables. The *lgg* of c_1 and c_2 is the following clause:

```
c:  p(X,W)  ←  q(Z,f(X)),s(t(c),W)
```

The inclusion relation in $\mathcal{X}$ and hence the generality relation in $\mathcal{H}$ are clearly of fundamental importance for induction. Indeed, it is natural to want to extend a subset of $\mathcal{X}$ that does not cover a positive instance and, conversely, to shrink a subset if it erroneously covers a negative instance. In terms of the hypothesis space, one needs either to generalize or to specialize an unsatisfactory hypothesis.

[4]The "typewriter" font is often used in the machine learning context for logical formulas and predicates.

The inclusion relationship in $\mathcal{X}$ induces a generality relation in $\mathcal{H}$, but this correspondence is actually used in the opposite direction: we work from the generality relation in $\mathcal{H}$ and derive inclusion in $\mathcal{X}$. Indeed, the whole point of using a hypothesis space is to be able to manipulate expressions in $\mathcal{L}_\mathcal{H}$ directly, to produce new hypotheses and therefore to explore $\mathcal{H}$. The question then becomes: what (easily carried out) syntactic manipulations in $\mathcal{L}_\mathcal{H}$ correspond to an inclusion relation in $\mathcal{X}$? In other words, what kind of syntactic *operator* on $\mathcal{L}_\mathcal{H}$ is guaranteed to be associated with an inclusion relation in $\mathcal{X}$ when used on an expression in $\mathcal{H}$? It turns out that this question can become quite tricky, depending on the hypothesis language that one is using. In the following example we will illustrate this point in the framework of a hypothesis language based on propositional logic.

EXAMPLE

Let us suppose that the description language of the hypothesis space is constructed from conjunctions of attribute values in a Boolean representation. For instance, `red ∧ noisy ∧ fast` could be a description of a `car` (where the symbol ∧ is to be understood as a logical conjunction). The expression `red ∧ fast`, obtained through the operation of *dropping a conjunct* (here `noisy`), corresponds to a more general hypothesis, since its associated coverage in $\mathcal{X}$ encompasses the coverage of `red ∧ noisy ∧ fast`.

In the language of propositional logic it is easy to check that the operator "drop a conjunct" is a generalization operator. Many other such syntactic operators, that are similarly associated with generalization (or conversely specialization) relations, are known in propositional logic. It is important to note that usually these operators do not allow all existing inclusion relations in $\mathcal{X}$ to be generated. In an analogous way, the language $\mathcal{L}_\mathcal{H}$ does not usually allow one to describe every subset of $\mathcal{X}$. The hypothesis language and its associated syntactic operators thus imply a bias on what is expressible within $\mathcal{H}$. As noted before, this is a necessary limitation if induction is to be possible at all; it may carry the price of having an ill-tuned language for the universe at hand.

5.1.5 Learning in a hypothesis space with a generality relationship

Once a generality lattice is available on $\mathcal{H}$, induction can be tightly controlled and therefore made efficient. As was suggested earlier, one approach would be to start with some hypothesis in $\mathcal{H}$, possibly generated at random, and then to use operators to produce new, more general (more specific), hypotheses as long as positive instances (negative instances) are not covered (are covered) by the

current candidate hypothesis. Indeed, this is what Tom Mitchell considered as a possibility when he started his Ph.D. in 1976, working on ways to learn in the framework of expert systems (Mitchell and Schwenzer, 1978). However, he soon realized that a more systematic method was possible and to be preferred.

Let us start with a hypothesis that is just a maximally specific generalization of a given example. This can be made very easy if the description language of the examples is part of the hypothesis language, $\mathcal{L}_{\mathcal{X}} \subset \mathcal{L}_{\mathcal{H}}$. In this case (known as the *single representation trick*), the maximally specific generalization of any example is unique and is just this very example. In any other case, the principle is to consider the other positive instances of the training set and, each time any example is not covered by a candidate hypothesis, a generalization operator is applied so as to produce the maximally specific generalization $msg(h_t, \vec{\mathbf{x}}_{t+1})$ of the current hypothesis, h_t, and the positive instance currently considered, $\vec{\mathbf{x}}_{t+1}$. In fact, at least in principle one could envision the possibility of generating in a breadth-first strategy *every* element of $\mathcal{H}$ that is part of $msg(\mathcal{P})$, where $\mathcal{P}$ is the subset of all the positive instances in the training set $\mathcal{S}_L$. Indeed, if the training data is not noisy (i.e., corrupted by description errors), if the language $\mathcal{L}_{\mathcal{X}}$ is sufficiently complete for the learning task, and if the hypothesis language $\mathcal{L}_{\mathcal{H}}$ is adequately biased then one can be certain that the target concept is at least as general as an element in $msg(\mathcal{P})$ (also called the $S\text{-}set$; see below).

Representation trick

Indeed, the target concept must at least cover every positive training instance. Of course, it could be more general than the elements in $msg(\mathcal{P})$ but it should not, if possible, cover negative instances of the training set. All generalizations of $\mathcal{P}$ must therefore not be so general as to cover any element of the subset $\mathcal{N}$ of all negative training instances. Let us call $mgg(\mathcal{P})$ (also called the $G\text{-}set$) the set of hypotheses that both cover all positive training instances and are maximally general without covering any negative training instances. Then all hypotheses that are both more general than at least one element of $msg(\mathcal{P})$ and more specific than at least one element of $mgg(\mathcal{P})$ are potential candidate hypotheses to approximate the unknown target concept. This set is called the *version space* $VS_{\mathcal{H}}(\mathcal{S}_L)$ associated with the training set.

Version space

Besides being one of the first to point out this view of supervised inductive learning, Tom Mitchell made a further contribution by discovering that the version space could be bounded by two sets when a generality relationship is available that guarantees the convexity of $\mathcal{H}$. These are, on one hand, the so-called *S-set*, the set of all hypotheses that are maximally specific generalizations of the positive instances $\mathcal{P}$ and do not cover any negative instance and, on the other hand, the so-called *G-set*, the set of all maximally general generalizations of $\mathcal{P}$ that do not cover any negative instance. Every hypothesis of the version space must lie between these two bounds, i.e., it must be more general than at

least one element of the *S-set* and be more specific than at least one element of the *G-set*.

Tom Mitchell provided an algorithm, the *candidate elimination algorithm*, that was able to maintain the *S-set* and the *G-set* while the elements of the training set were given in an incremental manner to the learner (Mitchell, 1982). However, even given this ingenious idea, of defining the whole space of solutions simply by maintaining the *S-set* and the *G-set*, this is not always practical because of the sheer size these bounding sets sometimes have. One is then condemned to explore only a subpart of the hypothesis space at any one time. Still, it is of interest to organize this search by either moving from the most specific hypotheses to more general ones, until one must stop in order to avoid covering negative training instances (the so-called *specific-to-general* or *bottom-up* strategy), or by moving from the most general hypotheses and specializing them in order to exclude negative instances while still covering the positive ones (the so-called *general-to-specific* or *top-down* strategy).

Candidate elimination algorithm

Most learning algorithms that exploit generality relations in the hypothesis space $\mathcal{H}$ are implementations of either of the above strategy or even of a combination of both, such as the *candidate elimination algorithm* (Mitchell, 1977).

5.2 Representation languages

There are not many representation languages for which generalization and specialization operators are known. The languages most used in machine learning are *propositional logic*, *first-order logic*, and *grammars*.

Languages for machine learning

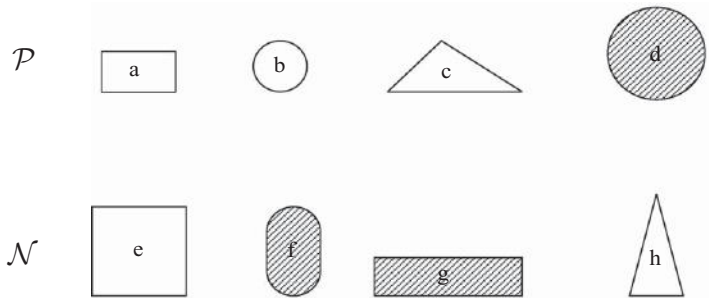
These representation languages are important in machine learning and, more widely, in artificial intelligence since they have three desirable properties.

1. *A generality relation exists* that guides the search in the hypothesis space for supervised learning.

2. They offer a *natural interface with prior knowledge*. More specifically:

   - they produce expressions (hypotheses) that are readable and generally easy to interpret, in contrast with, for instance, neural networks or support vector machines (SVMs),
   - they allow one to (easily) incorporate domain knowledge to help learning, for instance under the form of rules.

3. They have an expressive power that allows them (in particular, first-order logic and grammars) to *express complex relationships* in the world.



Attributes of the learning events {a, b, c, d, e, f, g, h, ...}
Shape : {square, rectangle, triangle, circle, ellipse}
Hatched : {YES, NO}
Length : real
Height : real
Classes $\{\mathcal{P}, \mathcal{N}\}$

Figure 5.7 Examples (*a–h*) of propositional learning events. Class $\mathcal{P}$ is defined to contain *a–d* and class $\mathcal{N}$ is defined to contain *e–h*.

There exists a very large literature about these representation languages. This section just aims at providing the basic notions that are necessary to understand the rest of the book.

5.2.1 Propositional representation

A language representation offers support for the *representation of facts and rules* about the world, as well as for *reasoning facilities* enabling one to infer new facts and rules from the current state of knowledge and, possibly, from new information gathered from the world. One of the simplest representation languages, where representation and reasoning can be well defined and be specified with a truth theory, is *propositional logic*.

Propositional logic

The simplest case considered in concept learning occurs when every *learning event* (also known as a *learning example* or *training example*), i.e., the arrival of some perceptual data from the external world, is a single item that is independent of other such items. Every event is characterized by a set of properties or attributes, which can be qualitative, such as `shape` and `color`, or quantitative, such as `weight`, `density`, and so on. Some examples of learning events of this type are provided in Figure 5.7 where a two-class classification problem is reported.

Learning event	Shape	Hatched	Length	Height	Class
a	rectangle	NO	1.5	2.3	$\mathcal{P}$
b	circle	NO	1.5	1.5	$\mathcal{P}$
c	triangle	NO	3.0	1.5	$\mathcal{P}$
d	circle	YES	2.2	2.2	$\mathcal{P}$
e	square	NO	3.0	3.0	$\mathcal{N}$
f	ellipse	YES	1.5	3.0	$\mathcal{N}$
g	rectangle	YES	3.5	1.0	$\mathcal{N}$
h	triangle	NO	1.3	2.5	$\mathcal{N}$

Figure 5.8 Tabular representation of the attributes of the learning events illustrated in Figure 5.7.

The upper row contains instances of the positive class ($\mathcal{P}$) whereas the lower row contains negative instances ($\mathcal{N}$). The induction problem consists in finding a function discriminating the positive from the negative class, i.e., a function that can predict the class value when this is unknown.

Representation with $\langle attribute, value \rangle$ vectors

The most immediate representation for learning events of the type in Figure 5.7 is the so-called $\langle attribute, value \rangle$ representation, i.e., a list of pairs whose first item is the attribute name and second item is the attribute value. If, as frequently happens, all objects are characterized by means of the same attribute set then this representation form leads to a tabular encoding of the learning events, as is shown in Figure 5.8, which tabulates the learning events in Figure 5.7 according to their attributes.

This form of tabular representation is found naturally in many data-mining applications, where data comes from relational databases.

Concept descriptions as and/or Boolean expressions

Let us now consider the problem of selecting a class of functions (hypotheses) able to discriminate the positive from the negative instances. We can individuate three major classes of functions:

Classes of functions

1. Boolean functions
2. continuous functions
3. probability distributions

Boolean functions are typically used together with algorithms for learning rules, or decision trees, whereas continuous functions are typically encoded using

neural networks or other mathematical approximators. Probability distributions are continuous functions that are explicitly trained in order to approximate probabilities. As continuous functions and probability distributions are outside the scope of this book, we will restrict the discussion to Boolean functions.

Independently of the way in which they are encoded (by rules or by decision trees), Boolean functions can always be reduced to a disjunctive normal form (a disjunction of conjunctions) of conditions on the attribute values.

Referring to Figures 5.7 and 5.8, a possible definition of the class $\mathcal{P}$ is

$$\mathcal{P} :: \text{(Hatched} = \text{NO)} \wedge (\text{Height} \leq 2.3) \vee \text{(Shape} = \text{circle)}, \tag{5.8}$$

which perfectly discriminates the positive instances from the negative instances.

In general, a Boolean function is a mapping

$$A_1 \times A_2 \times \ldots \times A_n \rightarrow \mathcal{Y} \tag{5.9}$$

from the space defined by the attribute set to the space of possible concepts.

It is easy to verify that, in the case of real-valued attributes, the space of Boolean formulas is infinite, since the set of possible conditions that can be set on any real attribute is infinite. Nevertheless, even in the case of discrete values or categoric attributes, the size of the hypothesis space defined by the class of Boolean functions can be very large. However, effective algorithms exist that are able to find good approximate hypotheses.

Finally, we observe that the language in expression (5.8) uses notation typical of propositional logic. In fact, propositional logic provides a formal framework for handling Boolean functions that has been used by many authors.

Covering test in propositional logic

In Section 5.1.4 the notion of the *coverage* of a hypothesis h, namely $cover(h)$, was introduced. Figures 5.4 and 5.5 illustrated its links with the "more general than" relation (Definition 5.2) between hypotheses. Clearly, this notion is bound to play a fundamental role in learning; therefore, providing a procedure to compute the coverage of a hypothesis is central to the search for good hypotheses.

The computation of $cover(h)$ can be reduced to the repetition over all available examples of the following task:

> given a hypothesis h and an example e, does e verify h or, conversely, is h true on e?

Matching problem or covering test

The above task is called the *matching problem* or *covering test*.

As it turns out, the matching problem has an efficient solution in propositional logic. In order to go into more detail let us consider a rather general

DNF representation

form for representing a concept in propositional logic, namely, *disjunctive normal form* (DNF). A propositional formula h in DNF is a disjunction of conjunctions γ_i of assertions c_j specifying attribute values of the concept instances:

$$h = \gamma_1 \vee \gamma_2 \vee \cdots \vee \gamma_k, \qquad \gamma_i = c_1 \wedge c_2, \wedge \cdots \wedge c_{r_i} \qquad (1 \leqslant i \leqslant k).$$

Any other representation in the propositional framework can be translated into DNF with a time complexity linear in the number of assertions. Given the concept description h and an object o to be classified, we say that o is recognized (classified) as a positive instance of the concept if its own description verifies h. To this end, it is sufficient that the description of the object satisfies at least one of the conjunctions γ_i. The process of comparing the assertions in h with those describing o is called the *matching problem*.

—— EXAMPLE ——

We will consider the instances in Figure 5.7 and their representation as $\langle attribute, value \rangle$ vectors tabulated in Figure 5.8. Let h be the formula $h = \gamma_1 \vee \gamma_2$ with

$$\gamma_1 = (\text{Dashed} = \text{NO}) \wedge (\text{Height} \leqslant 2.3),$$
$$\gamma_2 = (\text{Shape} = \text{circle}).$$

If we consider example a in Figure 5.8, it can be written as
$a = (\text{Shape} = \text{rectangle}) \wedge (\text{Hatched} = \text{NO}) \wedge (\text{Length} = 1.5)$
$\wedge\ (\text{Height} = 2.3)$.
By comparing the formula h with the example description, we can see that γ_2 is *false* for a but γ_1 is *true*. Thus h covers a.

Considering example $g = (\text{Shape} = \text{rectangle}) \wedge (\text{Hatched} = \text{NO}) \wedge (\text{Length} = 1.3) \wedge (\text{Height} = 2.5)$, we see that neither γ_1 nor γ_2 are *true*. Thus, h does not cover g.

Now that the example above has given us an intuition of how the covering test can be effectively computed, we can describe how the matching problem can be approached, in general, within propositional logic.

Examples in $\langle attribute, value \rangle$ form can be translated into a set of *ground assertions*, each corresponding to an *atomic test*, that possibly occur in a hypothesis h. Thus an atomic test consists of checking the value of an attribute. The hypothesis h, plus the assertions obtained from an example e, represent a propositional theory. Resolution (see Section 5.2.3) can be used to check the consistency of the theory. If it derives *false* then h does not cover e; otherwise, h covers e.

We note that atomic tests are nothing other than Boolean variables, which assume the value *true* or *false* depending on the value of the corresponding attribute in e. The covering test for a hypothesis has a complexity that is linear in the number of attributes and in the number of instances to be classified. However, if the framework is extended to include full propositional logic, the covering test complexity may become exponential. For instance, this happens when concepts are defined not by a single rule but by means of a theory, which is constructed incrementally. In this case, logical deduction may need to construct a refutation tree having a size that is exponential in the number of clauses.

For the sake of simplicity, in this book we will always consider the simpler procedure described earlier (and applied in the example above) when performing a covering test.

5.2.2 Relational representation

In the previous sections concept learning was defined in the framework provided by propositional logic, where a concept corresponds to a propositional assertion on a single object, which may be true or false depending on the values of its attributes. Here we will discuss how concept learning can be extended to the more general framework of *first-order logic* (FOL). More specifically, a concept will be made to correspond to a relation involving several objects, for which specific requirements must be satisfied. An example of a concept is the relation *x is the boss of y*, which, in order to be verified, requires a set of other relations involving x, y, and possibly other entities to be verified as well. These could be, for instance, *x and y work in department z*, *x is the director of z*, and *y is a programmer*.

First-order logic

As we will see in the following, the extension to first-order logic is not trivial: its impact on the complexity of the learning task can be dramatic.

INDUCE, developed by Michalski (1983), represents the first attempt at developing a learner in first-order logic. Later, other programs adopting the same framework as INDUCE appeared; these include ML-SMART (Bergadano *et al.*, 1988) and FOIL (Quinlan, 1990). Finally, a radical boosting of the field occurred owing to the birth of the so-called inductive logic programming (ILP) approach (Muggleton and Feng, 1990; Muggleton, 1991), which was followed by a large output of publications.

The origin of FOL learning

5.2.3 The problem setting

In Section 5.2.1 we saw how, in the propositional setting, learning events can be represented as $\langle attribute, value \rangle$ vectors, collected into tables. Starting from the propositional framework we will now introduce the relational framework. More

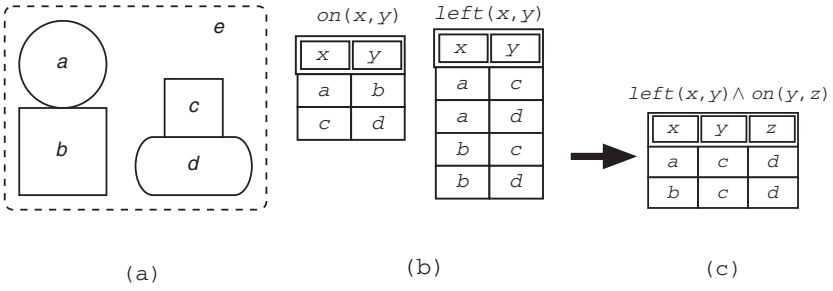


Figure 5.9 (a) A block-world instance *e*, composed of four objects, *a*, *b*, *c*, and *d*. (b) Tables describing *e* when the description language contains only two predicates, namely $on(x, y)$ and $left(x, y)$. (c) Substitutions for x, y, z satisfying the hypothesis $h(x, y, z) = left(x, y) \wedge on(y, z)$. More precisely, $h(a, c, d) = left(a, c) \wedge on(c, d)$ is true in *e*.

specifically, a learning event becomes a complex scenario where many objects may exist on which relations may be defined. In this framework concept learning becomes equivalent to learning the definition of a new relation between the objects in the scenario and the hypothesis space becomes the space of possible well-formed first-order formulas.

Data representation as multiple $\langle attribute, value\rangle$ vectors

Structured objects

A *structured* object is a set of elementary objects for which interdependency relations may exist. Consider, for instance, the block-world scenario in Figure 5.9(a), in which there are four elementary objects. Each object is characterized by a vector of attributes describing its specific features. Moreover, two relations (or predicates) between object pairs are defined: $left(x, y)$, and $on(x, y)$.

Extending the framework of Section 5.2.1, this scenario may be represented as a set of tables, one table for describing the objects and another for every relation (see Figure 5.9(b)). Notice that in the given example all objects are of the same type and are described by means of the same attribute vector. In general, however, the objects may have different types and so be described by different attribute vectors. Then a table needs to be given for each type.

In this new framework, to extend the notion of a concept (class) introduced in the propositional framework is immediate.

Concepts

Definition 5.3 {***Concept***} A concept is an *n*-ary relation $c(x_1, x_2, \ldots, x_n)$, where $x_1, x_2, \ldots, x_n$ denote generic objects. Let e be a composite object (*a scenario*), containing a set of elementary objects $O = \{o_1, o_2, \ldots, o_m\}$. Every *n*-tuple $\langle o_1, o_2, \ldots, o_n\rangle$ built on O for which $c(o_1, o_2, \ldots, o_n)$ is true is a positive instance of the concept *c*.

The relations $on(x, y)$ and $left(x, y)$ in Figure 5.9 can be considered as elementary binary concepts. Moreover, new concepts can be defined starting from them. For instance, $h(x, y, z) \equiv on(x, y) \wedge left(y, z)$ is a new ternary concept defined in terms of the two basic concepts (see Figure 5.9(c)).

In the same way, other concepts of different arities can be defined. For instance $h_1(x, y) \equiv \exists z[on(x, y) \wedge left(x, z) \wedge left(y, z)]$ is a binary concept, whereas $h_2(x) \equiv \exists y, z[on(x, y) \wedge left(x, z)]$ is a unary concept.

It is immediate to verify that the table describing the concept $h(x, y, z)$ defined above can be obtained by means of the natural join between the tables describing $on(x, y)$ and $left(x, y)$. Moreover, the tables $h_1(x, y)$ and $h_2(x)$ can be obtained from the natural join of $on(x, y)$ and $left(x, y)$ followed by projections onto x, y, and x, respectively. Every row in the new tables corresponds to a concept instance.

The Horn clause representation language

In order to deal with relations between objects, an adequate language is necessary. As already introduced above, relations may be described in an abstract way through predicate calculus. As an example, let us consider the expression $on(x, y)$. The symbol $on(x, y)$ denotes a predicate with an associated table; x and y are variables ranging over all the objects existing in the scenario provided by the learning event. When the values of x and y correspond to two items occurring in the same row in the table associated with predicate, the predicate is true; otherwise it is false.

Restricted predicate calculus

Thus predicate calculus provides a powerful tool for describing relational concepts. However, unrestricted predicate calculus is at the same time too powerful and too complex for this purpose. In practice, Horn clause languages (Kowalski, 1979) (a restricted predicate calculus) are used (Muggleton, 1992).

Before introducing Horn clauses we need to introduce logical clauses. *A logical clause* is a disjunction of atomic assertions, where an *atomic assertion*, for which the term *literal* is used in the following, is a possibly negated predicate of arity n ($n \geqslant 0$), i.e., it is applied to n arguments. In the more general case an argument may be a *constant* (the name of an item), a *variable*, or a *function*. An example of a clause is provided by the logical expression

Clauses

$$p(x) \vee q(x, y) \vee \bar{r}(y, z) \vee \bar{s}(z, y). \tag{5.10}$$

Clauses are implicitly considered as universally quantified, i.e., expression (5.10) can be rewritten equivalently as

$$\forall_{x,y,z}\ [p(x) \vee q(x, y) \vee \bar{r}(y, z) \vee \bar{s}(z, y)]. \tag{5.11}$$

In other words, a clause is true if it is true for any value that can be assumed by the variables occurring in it.

Horn clauses

Clausal representation plays a fundamental role in automated deduction. On the one hand, any set of logical equations in first-order predicate calculus can be translated into clausal form (see, for instance (Kowalski, 1979)). On the other hand, there exists a simple and complete deduction rule, namely *resolution* (Kowalski, 1979), which is suitable for implementation in automated reasoning systems. However, the logical languages used in automated reasoning are usually based on *Horn clauses*; as mentioned above these are a restricted form of clausal representation. A Horn clause is a clause in which at most one literal is not negated. An example of Horn clause is the logical expression:

An equivalent notation for Horn clauses, easier to compile for an automated program, consists in replacing the symbol $\wedge$ with a comma, and the symbol $\leftarrow$ with :-.

$$p(x) \vee \bar{q}(x,y) \vee \bar{r}(y,z) \vee \bar{s}(z,y). \tag{5.12}$$

By remembering that a logical expression of the type $\bar{p} \vee q$ is equivalent to $p \rightarrow q$, where the symbol $\rightarrow$ denotes a material implication, expression (5.12) can be rewritten as:

$$p(x) \leftarrow q(x,y) \wedge r(y,z) \wedge s(z,y), \tag{5.13}$$

which is the most popular format for Horn clauses. For such clauses, the resolution inference rule can be simplified with respect to its general form, leading to more tractable implementations. Moreover, the resolution rule can be "inverted" (*inverse resolution*, see Muggleton, 1992; Muggleton and Buntine, 1988), providing an automated induction schema. For the above reasons, Horn clause languages are those most frequently used in relational learning and inductive logic programming. An attractive property of Horn clause languages is that they offer a homogeneous framework for describing learning instances, background knowledge, and concept definitions.

Multiple vector representations are mapped to sets of ground assertions

Multiple vector representation

We have seen how structured objects can be described by means of a set of relational tables (see Figure 5.9). It is immediate to represent tables as sets of ground Horn clauses, i.e., clauses whose terms are only constants: every row in a table is mapped into a ground clause. As an example, the tables headed $on(x,y)$ and $left(x,y)$ in Figure 5.9 can be described by a set of ground clauses as follows:

$$\begin{aligned}
&on(a,b) \leftarrow true,\\
&on(c,d) \leftarrow true,\\
&left(a,c) \leftarrow true,\\
&left(a,c) \leftarrow true,\\
&left(b,c) \leftarrow true,\\
&left(b,d) \leftarrow true,
\end{aligned}$$

where every table row defines a ground assertion stating that the specific relation is $true$ for the tuple of objects in the row.

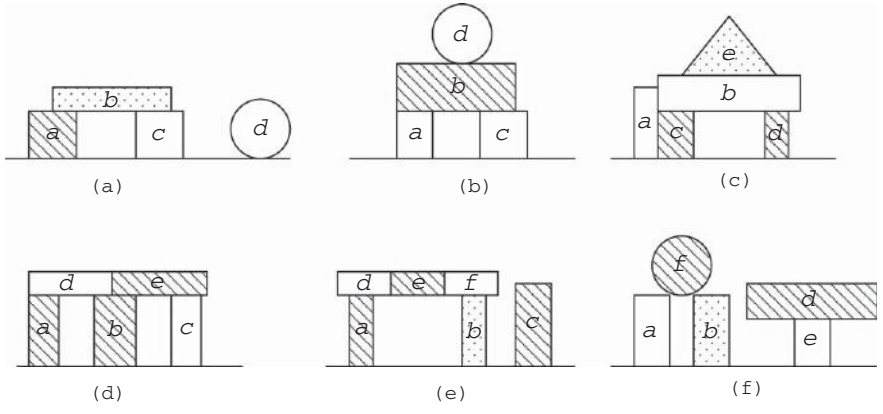


Figure 5.10 Six relational learning events, each based on a combination of objects labeled a, b, ...

Concept definitions are mapped to sets of clauses

In Section 5.2.1 we discussed how concepts can be defined in terms of and/or Boolean expressions. It is now immediate to verify that any form of Boolean expression can be rewritten in terms of Horn clauses in propositional logic. For instance, expression (5.8) can be equivalently written as

$$\mathcal{P} \leftarrow (\text{Hatched} = \text{NO}) \wedge (\text{Height} \leq 2.3),$$
$$\mathcal{P} \leftarrow (\text{Shape} = \text{circle}).$$

The same formal structure can be used in first-order logic to provide definitions of relations. A conjunctive definition can be stated through a single clause. For instance, considering the example in Figure 5.9, a new relation $\mathit{left\text{-}on}(x)$ can be defined by means of the clause

$$\mathit{left\text{-}on}(x) \leftarrow \mathit{on}(x, y) \wedge \mathit{left}(y, z). \tag{5.14}$$

The meaning of (5.14) is that any object x for which there exists another object y such that x is on y and for which there exists a third object z such that y lies to the left of z satisfies the relation $\mathit{left\text{-}on}$.

The more powerful framework offered by first-order logic allows different forms of concepts to be represented, corresponding to learning tasks of different complexity. We will discuss this point using a more complex example, shown in Figure 5.10, where six different scenarios are provided. Each scenario is a specific learning event containing instances of both elementary and composite concepts.

The simplest form of a relational concept is one where the concept is a 0-arity relation, which may be true or false in a given scenario depending on the relationships between the elementary objects it contains. Referring to Figure 5.10, an example of 0-arity concept may be the assertion "*an arch is present*". To decide whether this assertion is true or false in a specific scenario, we must verify whether it contains a set of objects realizing an arch. A possible definition for this concept could be:

$$\text{“}an\ arch\ is\ present\text{”} \\ \leftarrow grounded(x) \wedge grounded(y) \wedge on(z,x) \wedge on(z,y). \quad (5.15)$$

We recall that variables occurring in the head (left-hand side) of a Horn clause are universally quantified, whereas those occurring only in the body are existentially quantified. Thus definition (5.15) will be true of a given scenario if it contains at least three objects such that, when substituted to for the variables x, y, and z, verify the conditions stated in the body. Referring to Figure 5.10, we can see that in scenarios (a), (b), (c), (f) there is a unique triple of objects satisfying definition (5.15). In scenario (d), the definition is verified (satisfied) by two different triples (a, b, d) and (b, c, e), whereas in scenario (e) definition (5.15) is not verified because no single object is simultaneously on both a and b. In the following we will return to this point and discuss how to exploit the first-order framework to deal with the more general arch instance in (e).

In the case of relations with arity greater than 0, concept definitions are more informative. More specifically, the components that realize the concept instance can be explicitly identified. For instance, the concept definition

$$arch(x,y,z) \leftarrow grounded(x) \wedge grounded(y) \wedge on(z,x) \wedge on(z,y) \quad (5.16)$$

explicitly identifies the three components of the arch. The advantage is that, when a scenario contains more than one instance, all can be identified. As an example, in scenario (d) of Figure 5.10, definition (5.16) will fit the two instances $arch(a, b, c)$ and $arch(b, c, e)$. The consequence is that the implicit assumption made by many learning algorithms that a scenario either provides a positive or a negative example of the target concept no longer holds. In principle, a scenario may provide many positive examples of the target concept. Moreover, the notion of a negative example is defined in a different way: any substitution of the variables in the target relation by object names that do not satisfy the body of any clause of the concept definition is implicitly a negative example. For instance, in scenario (a) of Figure 5.10 the triples (a, c, d), (b, c, d), (a, b, d), etc., are negative instances of the concept $arch(x, y, z)$, according to definition (5.16).

Positive and negative examples

Since negative concept instances grow combinatorially with the number of elementary objects in a scenario, they are implicitly defined making the so-called

closed-world assumption (Kowalski, 1979): any instance not explicitly declared *positive* is to be considered as *negative*.

Therefore, dealing with n-arity relations requires learners more powerful than those suitable for 0-arity concepts. In fact, such learners must be able to deal with the presence of multiple instances in a single scenario and with the closed-world assumption.

Let us consider again the problem of learning concept definitions involving a variable number of objects. It can be solved by means of recursive definitions. Referring to the learning events given in Figure 5.10 and setting a 0-arity target concept, a suitable definition could be:

$$
\begin{aligned}
&\text{“}an\ arch\ is\ present\text{”} \leftarrow grounded(x) \wedge grounded(y) \\
&\qquad\qquad \wedge on(z,x) \wedge on(w,y) \wedge connected(z,w), \\
&connected(x,x) \leftarrow T, \\
&connected(x,y) \leftarrow adjacent(x,y), \\
&connected(x,y) \leftarrow connected(x,w) \wedge adjacent(w,y).
\end{aligned}
$$

Here the girder of the arch is defined as a sequence of connected objects: two objects x and y are connected if they are adjacent or if there is a chain of connected objects adjacent to both of them. Moreover, an object is connected to itself. It is immediate to verify that this definition will be satisfied in scenario (e) of Figure 5.10 also. However, learning recursive definitions such as that described above dramatically increases the size of the hypothesis space and requires smarter algorithms.

A further requirement is to extend the framework to handle n-ary relations when the objects involved in a concept are complex and contain a variable number of elementary components. For instance, referring to the concept $arch(x,y,z)$, we are looking for a definition covering scenario (e) of Figure 5.10 also. Up to now we have considered only clauses in which the terms occurring in the predicates are variables or constants. In other words, we have used a *function-free* description language, i.e., a DATALOG language.

In order to deal with composite objects we will need to use functions, thus increasing the power of the language. Referring to our simple case study, we will define a complex object by means of the composition function $link(x,y)$, which constructs a new object connecting two simpler objects x and y. Therefore a revisited definition of $arch(x,y,z)$ could be

$$
\begin{aligned}
&arch(x,y,z) \leftarrow grounded(x) \wedge grounded(y) \wedge on(z,x) \wedge on(z,y) \\
&\qquad\qquad \wedge girder(z), \\
&girder(x) \leftarrow object(x), \\
&girder(link(x,y)) \leftarrow adjacent(x,y) \wedge object(x) \wedge girder(y).
\end{aligned}
$$

Applying this definition to scenario (e) in Figure 5.10 will return the instance $arch(a, b, link(d, link(e, f)))$.

However, the leap to non-DATALOG languages further increases the complexity of the learning task and, consequently, of the learner. In particular, as we will discuss below, the most frequently used rule for searching a hypothesis space, i.e., θ-subsumption, becomes incomplete for non-DATALOG languages.

Covering test in first-order logic

In Section 5.2.1 we saw that the covering test in propositional logic can be defined either as a deduction task in a propositional theory including both the hypothesis and the example description or as a task directly matching the assertions in the hypothesis and the description of the example.

The deductive view can be immediately extended to the relational (first-order logic) framework. This was in fact the initial attempt in relational learning (Quinlan, 1990). The idea of learning that uses relational theories is very appealing and the deductive paradigm for implementing the covering test is sound within this framework. Nevertheless, in general it is too complex and becomes rapidly intractable as the complexity of the theory increases.

Therefore, a restricted form of deduction, called θ-subsumption (Plotkin, 1970), is used in practice (Muggleton and De Raedt, 1994). In the following we will provide both a formal definition of θ-subsumption and an intuitive explanation of how it works; it was introduced for the first time by Plotkin (1970). In order to provide a definition of θ-subsumption, we need to recall that a FOL formula may contain as arguments constants, variables, and/or functions; these are collectively known as *terms*. Then, a generic formula can be written as $\varphi(t_1, t_2, \ldots, t_n)$, where the t_i $(1 \leqslant i \leqslant n)$ are terms. Formally:

Definition 5.4 $\{$***θ-subsumption***$\}$ Let $\varphi(t_1, t_2, \ldots, t_n)$ and $\psi(s_1, s_2, \ldots, s_m)$ be two first-order logic formulas. We will say that $\varphi(t_1, t_2, \ldots, t_n)$ *subsumes* $\psi(s_1, s_2, \ldots, s_m)$ if there exists a renaming θ of the terms in φ and ψ that transforms φ into a sub-formula of ψ.

As already mentioned, in this book only the DATALOG subset of first-order logics will be considered. In DATALOG, which is a function-free language, terms are only allowed to be either variables or constants. In particular, a hypothesis h is a conjunction of predicates containing variables, whereas an example e is a conjunction of *ground literals*, which are either constants denoting the names of the components of e or values of the attributes characterizing these components. Then, the definition of θ-subsumption can be simplified.

Definition 5.5 $\{$***θ-subsumption in DATALOG***$\}$ Let $h(x_1, x_2, \ldots, x_n)$ be a first-order logic hypothesis, with variables x_i $(1 \leqslant i \leqslant n)$ and let e be an

example. We will say that h subsumes e if there exists a substitution θ for the variables in h that makes h a subset of e.

Subsumption and generality

Notice that if $\varphi(t_1, t_2, \ldots, t_n)$ and $\psi(s_1, s_2, \ldots, s_m)$ are two DATALOG formulas, saying that φ subsumes ψ implies that $\varphi \succcurlyeq \psi$, i.e., that φ is more general than ψ. Moreover, if h subsumes e then h can be obtained from e by dropping some predicates and turning some constants into variables in e.

In general, given two formulas φ and ψ, the θ-subsumption between them is not univocally defined because more than one unification θ such that φ subsumes ψ may exist. Actually, in some cases it may be interesting to count the number of alternative ways in which θ-subsumption can be instantiated.

The θ-subsumption relation offers a means to perform the covering test between a hypothesis h and a learning event e or even to test the *more-general-than* relation. However, it is *per se* a combinatorial problem, as we will discuss later on.

EXAMPLE

Let us consider again the problem illustrated in Figure 5.9. Let $h_1(x, y, z) = \mathit{left}(x, y) \wedge \mathit{on}(y, z)$ and $h_2(u, v) = \mathit{on}(u, v)$ be two hypotheses. It is immediate to see that the substitution $\theta = \{u/y, v/z\}$ makes h_2 a subset of h_1. By looking at the table in Figure 5.9(c) one can see that h_1 is verified by the two tuples (a, c, d) and (b, c, d). If we project the table corresponding to h_1 onto the last two columns then we obtain a single pair (c, d), whereas the hypothesis h_2 is verified by the two pairs (a, b) and (c, d). This is a confirmation that $h_2 \succcurlyeq h_1$.

Let us now consider again hypothesis h_2 and the learning event e defined in Figure 5.9(a). We would like to test whether h_2 covers e. The learning event e can be written as

$$e = \mathit{on}(a, b) \wedge \mathit{on}(c, d) \wedge \mathit{left}(a, c) \wedge \mathit{left}(a, d) \wedge \mathit{left}(b, c) \wedge left(b, d).$$

By applying to h_2 the substitution $\theta = \{u/a, v/b\}$ we see that h_2 becomes a subset of e. However, the substitution $\theta = \{u/c, v/d\}$ has the same effect. Thus in this case there are two substitutions that prove that h_2 covers e.

An informal but intuitive way for testing whether a hypothesis h covers a learning event e is to consider each predicate in h as a *test* to be performed on e. This can be done by binding the variables in h to components of e and then ascertaining whether the selected bindings verify in e the predicates appearing in h. The binding procedure should be made for all possible choices of the objects in e. The procedure will stop as soon as a binding satisfying h is

found, thus reporting *true*, or it will stop and report *false* after all possible bindings have been explored. The procedure is called, as in the propositional case, "matching h to e". Matching h to e has the advantage that it avoids the need to translating the learning event e into a ground logical formula, as, usually, examples come in tabular form. In practice, several learners use this matching approach for testing coverage (Bergadano and Giordana, 1988; Quinlan, 1990; Botta and Giordana, 1993). Moreover, matching is much more practical when used in machine learning and data mining algorithms working on databases. However, matching and θ-subsumption testing are equally computationally demanding. Matching h and e will be the subject of Chapter 9.

A procedure for the matching problem

5.2.4 Sequence or string representation

A very important kind of datum, which is being actively investigated in the areas of machine learning and data mining, is represented by *sequences*.

A sequence is probably the most frequent form in which data are obtained from the environment. Sequences can be seen as the output of a sequential process producing item after item according to the logic that controls its behavior. Thus sequences can also be thought of as structured objects, where a total order is defined between the elementary objects that are output by the generative process. Therefore, in sequences it is usually assumed that an item depends directly or indirectly on the previous items and, in turn, can influence the following items but not vice versa. In this sense a sequence can be seen as a special case of a structured object. However, sequences can be unlimited whereas the structured objects considered in relational learning are usually finite.

In many cases the objects in the sequence are each characterized by a single attribute, which may be a real value or a symbol. In the first case they are referred to as *temporal series* and in the second case as *strings*. Examples of temporal series are the stock market index and the continuous signals produced by an analogic equipment, whereas examples of strings are written texts and DNA.

In the following we will briefly review the most frequently investigated supervised learning tasks occurring in sequences. More specifically, we will consider three tasks:

1. predicting a property or the value of an item on the basis of the past history;
2. identifying the process generating a sequence;
3. sequence tagging.

Then we will discuss the classes of functions useful for approaching these kinds of learning task.

Learning to predict a property of the next item
A typical example of task 1 could be that of learning to forecast the weather tomorrow on the basis of the weather sequence observed in the recent past. Assuming a limit on the length of the past sequence, defined as the part of the sequence that could significantly influence the prevision for the future day, this learning problem can be reduced simply to a problem of relational learning, where the days in the considered window in the past history are the elementary objects in an environment, like those considered in relational learning. Moreover, if the window in the history has a fixed size, it is not difficult to imagine how the problem could be further reduced to a problem of propositional learning: the window could be encoded as an attribute-value vector.

Learning to identify the generative process for a sequence
Many important problems related to sequence analysis can be formalized as a the multi-class classification problem: that of identifying the process that generates the observed sequence from a set of *known processes*. However, it is also worth mentioning a two-class variant of this task, consisting in deciding whether a sequence belongs to a known process. In this case the negative examples can be generated by any process different from the known process.

Examples of the multi-class problem are the recognition of isolated words and the identification of a user navigating on the web. An example of the two-class problem is the authentication of a user profile in fraud detection.

Depending on the structure of the sequences generated by different processes, the problem of discriminating between processes can present quite different levels of difficulty. In the simplest cases, discrimination can be made on the basis of single items or short subsequences specific to the various generative processes. In the general case, an abstract characterization of the global generative process may be necessary.

Learning to accomplish sequence tagging
Given a set $\mathcal{Y}$ of labels, the task of sequence tagging consists in assigning a label $y \in \mathcal{Y}$ to every item in the sequence. In general the label corresponds to an abstract category whose meaning depends on the specific application. In isolated-word recognition it could be a phonetic category, whereas in a DNA strand it could be the name of a protein. When the label of an item x_i can be assigned on the basis of the items immediately preceding x_i, the task can be reduced to simple prediction. Nevertheless, in the most general case a label may

correspond to some specific phase of the generative process to be recognized. Then, sequence tagging may entail a task of process identification.

The hypothesis space

Many different approaches to encoding hypotheses related to sequences have been proposed; they depend upon the specific task, the sequence format, and the errors that are possibly present. When the items are characterized as attribute vectors, prediction or tagging tasks can be undertaken using the approaches described for the relational–propositional framework.

However, very different approaches are used when sequences are simply strings of symbols. Depending on the degree of noise present in the strings, three main approaches can be distinguished:

1. *string matching*;
2. *formal language*;
3. *Markov chain*.

The string-matching approach has many variants, ranging from dynamic programming (Gussfield, 1997) to string kernels (Shawe-Taylor and Cristianini, 2004), which all share the same assumption, that a string s can be attributed to a specific generative process on the basis of a set of substrings of s.

In the formal language approach (Hopcroft and Ullman, 1969) it is assumed that a string is a sentence generated by an agent (automaton) speaking a specific language. Then, any question related to a string can be answered by parsing the string according to the grammar governing the language of the agent. The learning task to solve is that of reconstructing this specific grammar, which represents the concept to be learned. Thus the hypothesis space is that of formal languages (grammars), whose instances are the observed strings. In the literature two kinds of space have been actively investigated, those of *regular* languages and *context-free* languages.

Finally, the Markov chain approach can be seen as an extension of the formal language approach in a Bayesian probabilistic framework (Rabiner, 1989; Durbin *et al.*, 1998) that accounts for the noise present in the data.

In Chapter 11 we will consider the formal language approach, where interesting discontinuous phenomena have been found.

5.3 Comments

In this chapter we aimed to provide the reader with the basic notions of machine learning, sufficient to understand the rest of the book. Machine learning

is a field that includes a vast range of approaches, both numerical (neural networks, support vector machine, statistical learning methods, etc.) and symbolic (learning rules, decision trees, logical formulas, grammars, graphs, etc.). Here, we have concentrated on symbolic learning, in particular on learning rules (both in propositional and first-order logics) and grammars. Nevertheless, other types of learning approach will be outlined in Chapter 7, when they have connections with statistical physics.

Even though most approaches to machine learning, including the learning of *optimal decision trees*, involve **NP**-complete problems, heuristic versions thereof have been quite successful in solving relevant practical problems. This is especially true for propositional approaches, where examples are not structured and can be described via a set of attributes. On the one hand relational learning, as discussed in Section 5.2.2, is much more complex and advances in this area have been slower. On the other hand, owing to its very complexity relational learning (and grammar induction) offers the best opportunities for establishing links with statistical physics, even though on the surface the two fields may appear to be very far apart. These links will be illustrated in Chapters 9–11.

6

Searching the hypothesis space

Contents

In Chapter 5 we introduced the main notions of machine learning, with particular regard to hypothesis and data representation, and we saw that concept learning can be formulated in terms of a search problem in the hypothesis space $\mathcal{H}$. As $\mathcal{H}$ is in general very large, or even infinite, well-designed strategies are required in order to perform efficiently the search for good hypotheses. In this chapter we will discuss in more depth these general ideas about search.

When concepts are represented using a symbolic or logical language, algorithms for searching the hypothesis space rely on two basic features:

1. a criterion for checking the quality (performance) of a hypothesis;
2. an algorithm for comparing two hypotheses with respect to the generality relation.

In this chapter we will discuss the above features in both the propositional and the relational settings, with specific attention to the covering test.

6.1 Guiding the search in the hypothesis space

If the hypothesis space is endowed with the more-general-than relation (as is always the case in symbolic learning), hypotheses can be organized into a lattice, as represented in Figure 5.6. This lattice can be explored by moving from more general to more specific hypotheses (top-down strategies) or from more specific to more general ones (bottom-up strategies) or by a combination of the two. Both directions of search rely on the definition of suitable operators, namely, *generalization* operators for moving up in the lattice and *specialization* operators for moving down. These operators are very important because they allow the generality relation between two hypotheses to be checked *intensionally* (i.e., by looking at their syntactic structures) rather than *extensionally* (i.e., by looking at their coverage).

Once we have selected a particular kind of strategy for the exploration, we need some criterion to assess the worthiness of the hypotheses encountered. To this end we may use the loss function (5.2) or some other heuristic criterion. In general, searching for the optimum of this criterion (be it the loss function or any other) on the hypothesis lattice is exponential, since the number of nodes is exponential. Thus most learners use a *greedy* search, in which moves on the lattice are guided by some *local* optimality criterion. A search is then a path on the lattice connecting nodes that locally look the the most promising. Of course, greedy search does not guarantee that optimality is reached; usually the search ends up in some local extremum. However, greediness makes the search tractable.

We are now in a position to illustrate the main approaches proposed in the literature for searching a hypothesis space, both in the propositional and in the relational settings.

6.1.1 Greedy search guided by information gain

A popular method for navigating through the hypothesis space is to go top-down, starting from the most general hypothesis $\top$ of the lattice (induced by the more-general-than relation) and constructing more-and-more-specific hypotheses via specialization operators. In the propositional framework, this is the approach used for learning *decision trees* or logical *decision rules* in DNF form.

A widely used heuristic for selecting the hypotheses in the lattice is based on the *information gain*, which exploits the notion of entropy. The entropy of an inductive hypothesis h is defined as the uncertainty about the class to be assigned to an instance $\vec{\mathbf{x}}$ that satisfies h. Let $\mathcal{Y}$ be the set of labels denoting the possible

classes to which the examples in a set $\mathcal{X}$ could belong, and let y_i be one of these labels. The entropy $S(h)$ of hypothesis h is defined by

$$S(h) = \sum_{i=1}^{|\mathcal{Y}|} -p_i \log_2(p_i), \tag{6.1}$$

p_i being the probability that an example verifying h belongs to class y_i. The p_i are usually unknown, but they can be estimated via the empirical frequencies measured on the learning set. Notice that $S(h) = 0$ when h is true only for the examples of a single class y_k. In this case, $p_k = 1$ and $p_j = 0$ for any $j \neq k$.

Let us consider now two hypotheses h_1, h_2 such that h_1 is more general than h_2 in the hypothesis lattice. The information gain $IG(h_1, h_2)$ is defined as the difference in entropy between h_1 and h_2:

$$IG(h_1, h_2) = S(h_1) - S(h_2) > 0. \tag{6.2}$$

Intuitively, (6.2) provides a measure of the decrease in uncertainty between classes when one moves from a more general to a more specific hypothesis. Usually the new hypothesis h_2 is built up from the current hypothesis h_1 by adding to h_1 a single constraint, the one having the maximum information gain with respect to h_1. The search stops when a hypothesis with entropy 0 is reached. Then all examples have been correctly classified.

Under the assumption that the learning set $\mathcal{S}_L$ is consistent (i.e., there is no example with multiple classification) and that it does not contain noise (such as, for instance, some example misclassified by the teacher), the above strategy aims at finding a *consistent* hypothesis, i.e., one that covers all positive examples and does not cover any negative example.

EXAMPLE

Let us consider the construction of a decision tree from the examples (learning events) given in Figure 5.7. We have the following descriptions:

$$\begin{aligned}
d &= (\text{Hatched} = \text{YES}) \wedge (\text{Shape} = \text{circle}),\\
a &= (\text{Hatched} = \text{NO}) \wedge (\text{Shape} = \text{rectangle}),\\
b &= (\text{Hatched} = \text{NO}) \wedge (\text{Shape} = \text{circle}),\\
c &= (\text{Hatched} = \text{NO}) \wedge (\text{Shape} = \text{triangle}) \wedge (\text{Height} < 0.7),\\
f &= (\text{Hatched} = \text{YES}) \wedge (\text{Shape} = \text{ellipse}),\\
g &= (\text{Hatched} = \text{YES}) \wedge (\text{Shape} = \text{rectangle}),\\
e &= (\text{Hatched} = \text{NO}) \wedge (\text{Shape} = \text{square}),\\
c &= (\text{Hatched} = \text{NO}) \wedge (\text{Shape} = \text{triangle}) \wedge (\text{Height} > 0.7).
\end{aligned}$$

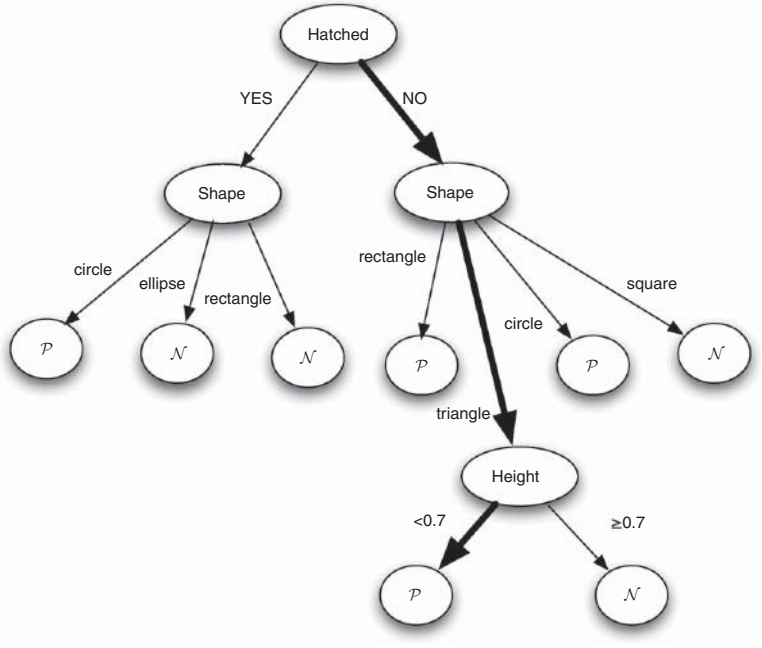


Figure 6.1 Decision tree classifying the examples (learning events) of Figure 5.7. The bold path corresponds to the hypothesis $h = (\text{Hatched} = \text{NO}) \wedge (\text{Shape} = \text{triangle}) \wedge (\text{Height} < 0.7)$. The entropy at the root $S(\top)$ is 1, because there are four positive and four negative examples in the learning set. The entropy of the right son of the root $S(\text{Hatched} = \text{NO})$ is 0.971, because the hypothesis (Hatched $=$ NO) covers three positive and two negative examples. Classification occurs by letting an example go down the tree until it reaches a leaf, where it receives the label associated with it. For instance, example c follows the bold path in the tree and is classified as positive.

A *decision tree* is a tree whose nodes correspond to attributes; the edges outgoing from a node ν are labeled with the values assumed by the attribute associated with ν. A hypothesis $h(\nu)$ is associated with each node ν; $h(\nu)$ is built up by concatenating the conditions found along the edges from the root to the node ν. "Within" each node ν are the examples covered by $h(\nu)$. A node is a *leaf* when all the examples within it belong to the same class.

In Figure 6.1 a decision tree perfectly classifying the examples (learning events) of Figure 5.7 is shown.

From a decision tree it is easy to compile a set of decision rules: it is sufficient to collect all the paths from the root to the leaves, and form rules that have the hypothesis on the leaf as body and the label associate to the leaf as head (classification). From the tree in Figure 6.1 the following rules can be obtained:

Decision trees can be immediately converted into propositional rules, and vice versa.

$$
\begin{aligned}
&(\text{Hatched} = \text{YES}) \wedge (\text{Shape} = \text{circle}) \rightarrow \mathcal{P},\\
&(\text{Hatched} = \text{NO}) \wedge (\text{Shape} = \text{rectangle}) \rightarrow \mathcal{P},\\
&(\text{Hatched} = \text{NO}) \wedge (\text{Shape} = \text{circle}) \rightarrow \mathcal{P},\\
&(\text{Hatched} = \text{NO}) \wedge (\text{Shape} = \text{triangle}) \wedge (\text{Height} < 0.7) \rightarrow \mathcal{P},\\
&(\text{Hatched} = \text{YES}) \wedge (\text{Shape} = \text{ellipse}) \rightarrow \mathcal{N},\\
&(\text{Hatched} = \text{YES}) \wedge (\text{Shape} = \text{rectangle}) \rightarrow \mathcal{N},\\
&(\text{Hatched} = \text{NO}) \wedge (\text{Shape} = \text{square}) \rightarrow \mathcal{N},\\
&(\text{Hatched} = \text{NO}) \wedge (\text{Shape} = \text{triangle}) \wedge (\text{Height} > 0.7) \rightarrow \mathcal{N}.
\end{aligned}
$$

Rule learning

Rules can be learned directly, as well, by using algorithms explicitly designed to do so (see for instance Dietterich and Michalski, 1983; Cohen, 1995). Rules are usually learned one at a time and, each time a rule is accepted by the learner, the set of examples covered by it is removed from the learning set. The acquisition of new rules stops when all positive examples are covered.

Limits of the greedy IG heuristic

Unfortunately, this simple strategy alone does not work well in most practical cases. First, it is not optimal because the greedy search strategy is incomplete; it does not take into consideration interactions between the constraints. In fact, it is possible that the conjunction of two or more constraints, which in isolation do not show the maximum information gain, constitutes a better hypothesis than that constructed by the above strategy. Second, it is quite rare that a consistent learning set is available. Inconsistencies may arise from errors at the time of data collection or may be due to an *a priori* misclassification on the part of the teacher who labels the examples or may be the effect of a poor choice of attributes, and so on. As a consequence, the learning set can contain examples that have the same description but different classifications. Even worse, the effect of noise on the attribute values can hide this problem, allowing two or more examples having in reality the same description to appear different only because of the noise. In these cases, the IG heuristic is likely to generate a disjunction of a number of very detailed hypotheses, each covering few examples (possibly only one). This phenomenon, well known in machine learning under the name of *overfitting*, produces hypotheses that are very good (if not perfect) on the training set but which show a poor classification ability on previously unseen instances (poor generalization power).

Pre-pruning

In order to combat this problem, the IG heuristic is usually combined with *pre-pruning* or *post-pruning* techniques based on other heuristics suggested by statistics or information theory. The idea of pre-pruning simply consists in requiring that a new, more specific, hypothesis h_2 not only exhibits the maximum

IG with respect to the current hypothesis h_1 but also exhibits a reduction in the classification error for instances not belonging to the training set $\mathcal{S}_L$. To this end a set of examples, called the *pruning* set, is kept apart and used for the sole purpose of pre-pruning (Breiman *et al.*, 1984). If this requirement is not satisfied then the specialization process stops and h_1 is added to the global solution.

Post-pruning

In the case of post-pruning the greedy search algorithm constructs a detailed (possibly overfitted) solution guided by the IG heuristic only. Then the solution undergoes a simplification procedure in which conditions not providing a significant reduction in the classification error of new learning instances are dropped.

Minimum description length

A powerful pruning heuristic is based on the *minimum description length* (MDL) principle (Rissanen, 1978). The MDL is a measure of complexity which counts the minimum number of bits necessary for describing a global hypothesis h that represents regularities in the data, plus the bits necessary to describe the data not explained by the hypothesis (the exceptions). According to this principle, a more specific hypothesis h_2 replaces the current hypothesis h_1 only if the MDL decreases from h_1 to h_2. Therefore, a detailed hypothesis covering only a few examples is likely to be rejected. The MDL principle has the advantage of not requiring a pruning set, and it is very easy to implement in pre-pruning techniques (Quinlan, 1986). However, it has two drawbacks. The first is that it is not computable, in the sense of computability theory; however, several good approximations of it are available. The second drawback is that the assumption that minimizing the MDL of a hypothesis also reduces the classification error on an independent *test set* $\mathcal{S}_T$ remains a conjecture to be proved. However, the method works well with decision trees, providing performances comparable to those ones obtained using a pruning set.

6.1.2 Lifting information gain to first order

We will now discuss the heuristics frequently used to guide learning algorithms in the relational framework. In this case also, many heuristics are based on the concept of information gain. However, in the relational framework, things are more complex than in the propositional framework, and the IG measure (6.2) cannot be applied directly as in the previous subsection.

As discussed in Section 5.2.3, a first-order formula can be satisfied in many different ways in a learning instance. Let us consider again the example of Figure 5.10 and the 0-arity concept:[1]

$$\text{“}arch\text{”} \leftarrow grounded(x) \wedge grounded(y) \wedge on(z,x) \wedge on(w,y) \wedge connected(z,w).$$

[1]Here we will make specific reference to 0-arity concepts because the analysis of phase transitions in relational learning is done using this type of concept.

Table 6.1 Numbers of alternative tuples satisfying h_1 and h_2 in the different scenarios of Figure 5.10; $\mathcal{P}$ is the class of positive instances

	(a)	(b)	(c)	(d)	(e)	(f)	total	$\mathcal{P}$
h_1	6	2	6	6	6	6	38	32
h_2	2	2	2	4	2	3	13	11

We saw that in scenarios (a), (b), (c), (f) there is a unique triple of objects satisfying this concept definition whereas in scenario (d) the definition is verified (satisfied) by two different triples of objects, namely (a, b, d) and (b, e, c). Finally, scenario (e) is a negative example of this simple definition, because no single object is simultaneously on both x and y. Suppose that we want to learn the concept "arch" using a general-to-specific greedy strategy guided by the IG heuristic. Suppose, moreover, that the hypothesis

$$h_1 \equiv \text{“}simple\ arch\text{”} \leftarrow grounded(x) \wedge grounded(y)$$

has already been constructed and that the algorithm is evaluating the IG with respect to the new formula

$$h_2 \equiv \text{“}simple\ arch\text{”} \leftarrow grounded(x) \wedge grounded(y) \wedge on(z, x)$$

obtained by adding the new literal (relation) $on(z, x)$ to h_1. Expression (6.2) will return $IG(h_1, h_2) = 0$, because both h_1 and h_2 cover all the learning scenarios. Nevertheless, this literal is the only possible choice for learning the correct definition. From this simple example it is clear that the heuristic widely used in propositional calculus does not work, without modification, in the relational framework.

Let us consider now the number of possible ways in which h_1 and h_2 are satisfied in the different scenarios in Figure 5.10. In Table 6.1 the numbers of alternative tuples of constants satisfying hypothesis h_1 and h_2, respectively, is reported.

We observe that the proportion of tuples in the positive scenarios is slightly higher for h_2 ($11/13 = 0.846$), than for h_1 ($32/38 = 0.842$). In fact, the literal $on(z, x)$ captures a pattern that is more frequent in the positive instances and is a necessary precondition for acquiring the correct definition. Therefore, new heuristic definitions of the information gain are proposed in relational learning; for each of h_1 and h_2 they depend on the ratio of the number of satisfying tuples in the positive examples and the total number of satisfying tuples.

In the remaining of this chapter we will illustrate the basic functioning and heuristics used by the machine learning algorithms in the experimental study reported in later chapters.

6.2 FOIL: information gain

The first relational learner proposed in the literature that was guided by a heuristic based on the information gain was FOIL (Quinlan, 1990; Quinlan and Cameron-Jones, 1993). This learner exploits a greedy hill-climbing search strategy, which is guided by the IG heuristic (6.2). FOIL was an extension to the relational case of the corresponding algorithm used in the learning of decision trees (Quinlan, 1986).

FOIL uses a top-down search strategy for building clauses that describe different modalities of the target concept or relation $c(x_1, \ldots, x_n)$. The basic version of the algorithm relies, for learning, on a database of tuples of constants, some labeled positive (i.e., they are in $\mathcal{P}$) and some negative (i.e., they are in $\mathcal{N}$). The algorithm consists of two nested loops. In the *outer loop* it iteratively constructs one Horn clause at a time; after a new clause is acquired, the covered positive learning tuples are removed from the learning set and the search restarts by looking for new clauses covering the remaining positive learning tuples. In the *inner loop* FOIL seeks a Prolog clause (see Section 4.15) of the form $c(x_1, \ldots, x_n) \leftarrow L_1, \ldots, L_k$, which characterizes some subset of the relation c. The clause is "grown" by starting with just the left-hand side and adding literals L_i $(1 \leqslant j \leqslant k)$ one by one to the right-hand side. This inner loop makes use of a local training set consisting of labeled k-tuples of constants extracted from the global training set.

The information gain heuristic plays its role in the construction of single clauses. Let h_1 denote the current hypothesis corresponding to a Horn clause $h_1 \leftarrow L_1, \ldots, L_i$. Moreover, let h_2 be a more specific Horn clause obtained by adding a literal L_{i+1} to $L_1, \ldots, L_i$. FOIL's information gain $IG(h_1, h_2)$ is given by the following expression:

$$IG(h_1, h_2) = t \log_2 \left(\frac{P_2}{P_2 + N_2} \right) - \log_2 \left(\frac{P_1}{P_1 + N_1} \right), \tag{6.3}$$

where P_1, P_2, N_1, N_2 are the numbers of tuples satisfying h_1 and h_2 in positive and in negative instances; t is the number of positive tuples covered by both h_1 and h_2. If h_2 is obtained by adding to h_1 a literal containing a new variable, a tuple covered by h_1 is considered to be covered also by h_2 if it is contained in a tuple covered by h_2.

FOIL also includes special heuristics in order to handle the case where no new hypothesis exists having a positive information gain with respect to the current one. In this case, FOIL adds a literal that extends the number of variables in the body of the clause.

6.3 SMART+: beam search

SMART+ is a multi-strategy learning system which incorporates a set of several alternative strategies that the user can select and/or combine, depending on the specific learning problem. Again, we will limit here the description of the system to its essence. The interested reader is referred to the work by Botta *et al.* (1993) for a complete description of the system.

Basically, SMART+ proceeds top-down as FOIL does. A distinctive feature is the use of *beam search* instead of pure hill-climbing. The maximum width w of the beam can be set by the user; for instance, if $w = 1$, beam search reduces to hill-climbing. A second feature is that SMART+ can learn multiple concepts at the same time. However, it is not able to learn relations of arity greater than 0. Finally, SMART+'s heuristic for guiding the search is more sophisticated than FOIL's and combines the information gain criterion with other criteria accounting for the completeness and consistency of the hypotheses and, possibly, for the available domain knowledge.

For the sake of simplicity we will restrict the description of the search heuristic to the case of the learning of only one positive concept (as with FOIL) without any domain knowledge.

Given a current conjunctive hypothesis h_1 (a Horn clause), any new hypothesis h_2 obtained by adding a literal to h_1's body (right-hand side) is evaluated by a function $\mu(h_1, h_2)$ that is the weighted sum of two terms:

$$\mu(h_1, h_2) = \alpha\mu_1(h_1, h_2) + (1 - \alpha)\mu_2(h_2) \qquad (0 \leq \alpha \leq 1). \tag{6.4}$$

The first term is the information gain of h_2 with respect to h_1 and is computed as follows:

$$\mu_1(h_1, h_2) = P_2 \log_2 \left(\frac{P_2}{P_2 + N_2} \right) - \log_2 \left(\frac{P_1}{P_1 + N_1} \right). \tag{6.5}$$

Expression (6.5) is analogous to (6.3), but with the difference that here the information gain is multiplied by the number of positive tuples satisfying h_2 instead of the number of tuples satisfying both h_1 and h_2. The second term has the following expression:

$$\mu_2(h_2) = \frac{|cover(h_2) \cap \mathcal{P}|}{|\mathcal{P}|} \; \frac{|cover(h_2) \cap \mathcal{P}|}{|cover(h_2) \cap \mathcal{P}| + |cover(h_2) \cap \mathcal{N}|}. \tag{6.6}$$

and is the heuristic rule used by a precursor of SMART+ (Bergadano *et al.*, 1988). More specifically, (6.6) is the product of two factors: the first is the ratio of the number of positive learning instances covered by h_2 and the total number of positive instances in the learning set, and is a measure of the completeness of h_2. The second factor is the number of positive instances covered by h_2 divided by the total number of instances (both positive and negative) covered by h_2, and is a measure of the consistency of h_2.

If α is set to 1, SMART+ uses only the information gain term in (6.4) while if α is set to 0 it uses the second term only. The default value is $\alpha = 0.5$.

Let B denote the size of the current hypothesis beam. At each step SMART+ evaluates, with rule (6.4), all the hypotheses that can be derived from those in the beam by adding one literal and retains the best B chosen from the union of the old and the new hypotheses. The search stops when all learning instances are covered or when any new hypothesis has a score lower than the worst in the beam.

6.4 G-Net: genetic evolution

Genetic algorithms are general purpose biologically inspired tools for performing a search in optimization problems (Goldberg, 1989). The basic idea consists in encoding tentative solutions to a problem as DNA chromosomes. Evolving the DNA according to the Darwinian paradigm of genetic evolution, the solutions tend to approach a global, or local, optimum. The literature of the field is very rich, and even an introduction to it is outside the scope of this book. The interested reader can find a good one in the book by Goldberg (1989).

As the task of concept learning can be cast as a search problem, genetic algorithms were proposed for this task very early on (Holland, 1986; De Jong *et al.*, 1993). Since then, many authors have proposed different evolutionary paradigms specifically tailored to concept learning. Here we will focus on G-Net (Giordana and Saitta, 1994; Anglano *et al.*, 1998), a task-oriented genetic algorithm that will be used in the following for investigating the emergence of phase transitions in relational learning. In fact, G-Net differs from other relational learners principally in its search strategy, which is based on a stochastic genetic search guided by the minimum description length (MDL) principle (Rissanen, 1978). As shown by Giordana and Saitta (1994), the portion of hypothesis space visited by G-Net is different from that visited by other learners; thus it can find solutions that others cannot find, and vice versa.

Like FOIL and SMART+, G-Net describes a concept as a set $\{h_1, h_2, \ldots, h_n\}$ of Horn clauses. Each clause is a partial definition covering a different modality of the concept. G-Net's inductive engine exploits a stochastic search

algorithm organized on two levels. The *lower level*, named the genetic layer (G-layer), searches for single Horn clauses h_i. The architecture of the G-layer derives from the diffusion model (Manderik and Spiessens, 1989) and integrates different ideas originating within the community of evolutionary computation and *tabu* search (Rayward-Smith *et al.*, 1989). The *upper level*, namely the supervisor, builds up a global concept definition φ out of the partially defined h_i's generated inside the G-layer, using a greedy set-covering algorithm.

From a computational point of view, the G-layer consists of a set of (virtual) elementary processors called G-nodes. Each G-node is associated with a single concept instance e^+ and executes a local evolutionary search aimed at covering e^+ as well as possible according to a fitness function based on the MDL principle. In more detail, the search starts with a maximally specific clause covering e^+, constructed by a *seeding* operator (Giordana and Saitta, 1994). Then this initial clause is evolved by applying a set of stochastic operators, which perform generalization, specialization, and random mutation. Hence, G-Net exploits a bidirectional data-driven search strategy. According to the MDL principle, a new clause obtained in this way receives a good evaluation when it is simple and consistent. Nevertheless, simple hypotheses also tend to be general and to cover many other examples; then the set of clauses φ' evolved by the G-layer tends to be highly redundant. As a consequence the basic task of the supervisor consists in extracting a minimal (according to MDL) set φ of clauses from φ'. Moreover, the supervisor interacts with the G-layer, focusing the search towards partial hypotheses covering positive examples not yet covered by the current set of clauses φ. This is done by increasing the number of G-nodes, searching the concept instances that are not covered or are poorly covered by the current clauses in φ.

6.5 PROGOL: exhaustive search

PROGOL is probably the best-known learning program developed in the ILP community and is available in two basic implementations, one in C language (C-PROGOL) and the other in Prolog (P-PROGOL).

The PROGOL algorithm was described in detail by Steve Muggleton (1995); a good introduction to the theory, implementation, and applications of ILP is provided by Muggleton and De Raedt (1994). P-PROGOL was intended to be a prototype for exploring ideas. It started in 1993 as part of a project undertaken by Ashwin Srinivasan and Rui Camacho at Oxford University. The main purpose was to understand the idea of inverse entailment, which eventually appeared in (Muggleton, 1995). C-PROGOL is an implementation written in C that contains its own Prolog interpreter and is aimed at increasing the computational efficiency. The main differences in implementation (other than language) are in

the search technique used and in the degree of allowed user interaction. During routine use PROGOL follows a very simple procedure, which consists of four steps.

1. *Select example* Select an example to be generalized. If none exists, stop; otherwise proceed to the next step.

2. *Build most specific clause* Construct the most specific clause that entails the example selected and complies with the language restrictions. This clause is usually a definite clause with many literals and is called the "bottom clause". This is sometimes called the *saturation* step.

3. *Search* Find a clause more general than the bottom clause. This is done by searching for some subset of the literals in the bottom clause that has the "best" score. Two points should be noted here. First, confining the search to subsets of the bottom clause does not actually produce all the clauses more general than the bottom clause, but this description is sufficient for a thumbnail sketch. Second, the exact nature of a clause's score is not really important here. This step is sometimes called the *reduction* step.

4. *Remove redundant examples* The clause with the best score is added to the current theory and all examples made redundant are removed. This step is sometimes called the "cover removal" step. Note here that the best clause may make clauses other than the examples redundant. Again, this will be ignored here. Return to step 1.

In other words the search starts bottom-up, as in G-Net, and the second step is similar to the seeding operator, where a maximally specific clause is constructed. Afterwards, more general clauses are built up using the inverse entailment operator. The basic search strategy used in this case is a best-first exhaustive search. Nevertheless, it can be modified by the user by supplying preference criteria and pruning techniques to limit the set of clauses generated, which otherwise could quickly become intractable.

As in the cases of FOIL and SMART+ the positive examples covered by a clause selected at the end of the third step are removed from the learning set before a new induction step is started.

6.6 Plateaus

When solving optimization problems, local search algorithms may be faster than complete search algorithms, but they tend to become trapped in local minima. An interesting phenomenon may occur during searching, i.e., starting from the

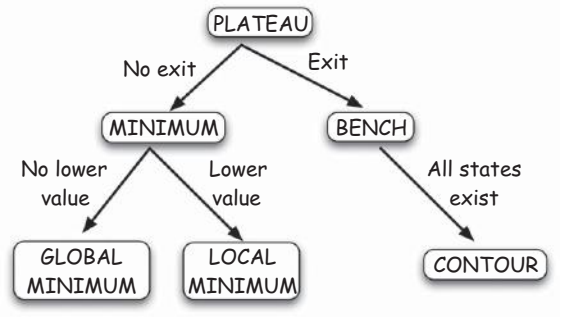


Figure 6.2 Classification of plateaus according to Frank *et al.* (1997).

current search state the algorithm cannot reach, in one step, any state that shows an improvement of the objective function. In other words, the algorithm has no guidance about where it should go next. A region in which the objective function does not change is called a *plateau*. In a plateau the search goes blind and turns into a random search until either an exit from the plateau is found or the search stops for some reason.

Frank *et al.* (1997) studied this phenomenon and defined different types of plateaus, as indicated in Figure 6.2. They used, for the experiments, the GSAT algorithm (Selman *et al.*, 1992) for solving a variety of 3-SAT problems. In the following description of their work we will assume that the objective function H is to be minimized. Plateaus are defined as any maximally connected region of the local search space over which the objective function is constant. They are divided, by the authors, into two classes: *local minima* and *benches*. We will now explain the difference.

Given a plateau, its *border* is the set of points in the search space that are neighbors to some state in the plateau but differ in the value of the objective function H. If the value of H in the plateau is lower than the value of H at any of its border points, then the plateau is a (global or local) *minimum*. If, on the contrary, the value of H at some point s in the border is lower than the value in the plateau then the points inside the plateau adjacent to s are called *exits*; plateaus with exits are called *benches*. Intuitively, minima are plateaus surrounded by regions of the search space where the objective function is greater than in the plateau. A greedy local searcher cannot escape from a minimum once it enters it. Benches are plateaus with exits towards regions where the objective function has lower values than in the plateau. Thus a searcher can escape from them. However, the experiments of Frank *et al*. showed that benches are often much larger than local minima; most have a high number of exits but some have very few. A bench may

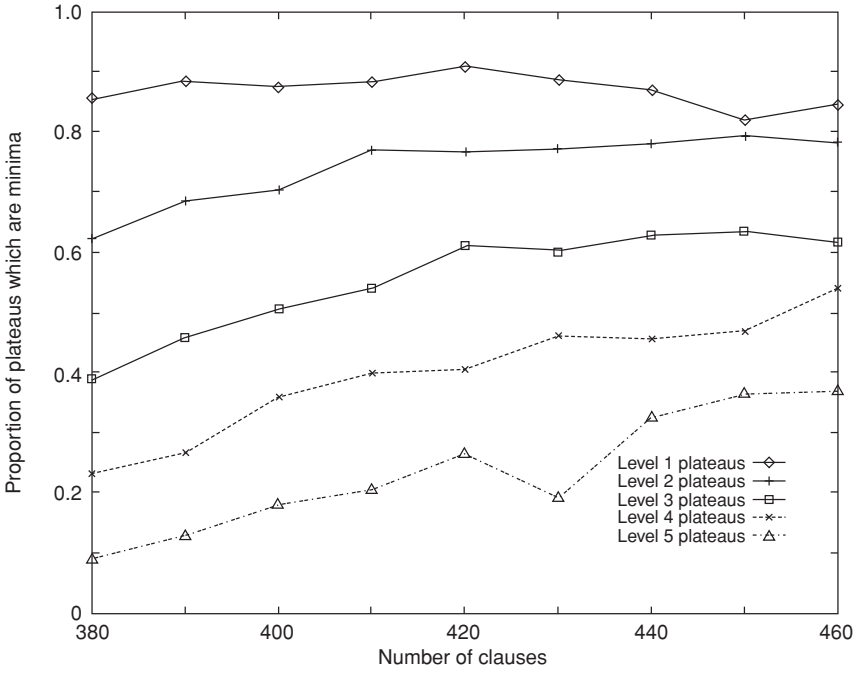


Figure 6.3 Proportion of minima among the plateaus found in 3-SAT by Frank *et al.* (1997). The levels correspond to the values of E. Reprinted from Frank *et al*. (1997) with permission.

consist entirely of exits; in this case it is called a *contour*, and a local searcher will visit only one point of such a bench before moving away from it.

Using the above definitions and applying the GSAT search algorithm (Selman *et al.*, 1992), Frank *et al*. (1997) investigated the behavior of plateaus in ensembles of 3-SAT problems with variable number $n = 100$, near the phase transition (see Chapter 3), for values of α in the interval $[3.8, 4.6]$. The objective function to be minimized was the number E of unsatisfied clauses. First, satisfiable problem instances were analyzed. In order to find a plateau, GSAT was run until it found a state with a prespecified level of E (from 0 to 5). Then, using a breadth-first strategy, all the states with the same value of E were searched for and recorded. At the same time the nature of the plateau (minimum or bench) was determined. The proportion of minima and benches in sets of 1000 problems for each value of E was computed. Figure 6.3 gives the results of this investigation. Another aspect investigated by the authors was the size distribution of the local minima and benches. For the same set of problems as before (100 variables

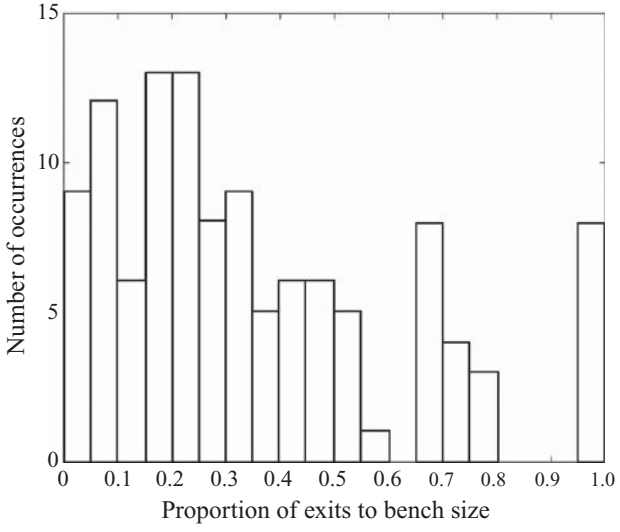


Figure 6.4 Number of benches with a given proportion of exits, for $E = 1$, in problems with 100 variables and 430 clauses. Reprinted from Frank *et al*. (1997) with permission.

and 430 clauses) and for $E = 1$ they found that the majority of local minima have size less than 300 states (the median was 48 states), but the distribution has a long tail, reaching to 10 000 states. In addition, by setting $E = 0$, global minima are obtained (note that the study was done for satisfiable instances); the set of global minima consists of several separate minima of widely varying size.

Analogous results were found for the bench size distributions; these again show long tails, but the benches are much larger (by a factor 10–30) than the minima and the size distribution tends to be flatter. As the difficulty of escaping from a bench depends on the number of exits, a relevant parameter to investigate is the ratio of the number of exits and the size of the bench. Figure 6.4 gives the experimental results. Finally, Frank *et al.* (1997) searched for features that could distinguish a satisfiable from an unsatisfiable problem instance, so that a local searcher could exploit this information early on. Unfortunately, nothing substantial was found.

Interesting results have also been found for the MaxSAT problem (Sutton *et al.*, 2009), where, in order to manage very large instances, it is fundamental to be able to navigate and escape plateaus effectively and to find exits where they exist. The authors modeled the presence of plateaus as a percolation process on

a hypercube graph and described how lower and upper bounds on a plateau's expected size can be estimated (together with their accuracy), given the E value of the plateau.

As we will see in later chapters, the presence of plateaus plays a fundamental role in relational learning as well.

6.7 Comments

The difference in complexity existing between representations of hypotheses in propositional and relational learning, described in Section 5.2, is matched by an analogous difference in the complexity of the algorithms devoted to acquiring the two kinds of knowledge. Globally, in order to solve a learning problem the learning algorithm, whatever its search strategy may be, must exploit some type of evaluation function to guide the search, as the exploration of the whole hypothesis space is out of question except in trivial cases.

There are basically three types of guidance that a learning algorithm can receive: a hypothesis evaluation function (e.g., the information gain or the minimum description length), training data, and domain knowledge. All three have advantages and drawbacks. Evaluation functions may be ineffective in some regions of the hypothesis space, owing to the presence of plateaus, but in other regions they may help the learner to find good hypotheses. Adhering too closely to training data, however useful, may mislead the search and generate overfitting, particularly in very noisy domains; on the other hand, training data may allow large parts of the hypothesis space to be disregarded. Finally, domain knowledge may help considerably in finding human-meaningful hypotheses, but it is difficult to handle.

Even though learning is a search problem, the machine learning community has developed specific algorithms rather than trying to use those available in the search community. One reason for this could be the feeling that learning is a special type of search, to which ad hoc algorithms will prove to be better suited than generic searchers. Some experimental studies support this opinion. Recently, as a consequence of the establishment of links with statistical physics, new efforts have been devoted to explore the possibility of using and developing new algorithms.

7

Statistical physics and machine learning

Contents

The study of the emergence of phase transitions, or, more generally, the application of statistical physics methods to automated learning, is not new. For at least a couple of decades ensemble phenomena have been noticed in *artificial neural networks*. In the first part of this chapter we will illustrate these early results and then move to currently investigated issues.

According to Watkin *et al.* (1993), statistical physics tools are not only well suited to analyze existing learning algorithms but also they may suggest new approaches. In the paradigm of *learning from examples* (the paradigm considered in this book), examples are drawn from some unknown but fixed probability distribution and, once chosen, constitute a *static quenched disorder* (Watkin *et al.*, 1993).

7.1 Artificial neural networks

Artificial neural networks (NNs) are graphs consisting of a set of nodes that correspond to elementary computational units, called "neurons", connected via

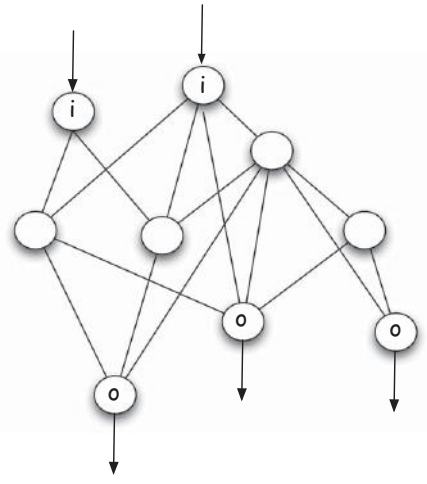


Figure 7.1 A generic structure of a neural network. Nodes labeled "i" are *input* nodes, nodes labeled "o" are *output* nodes, and blank nodes are *internal* nodes.

links recalling "axons" and "synapses". Even though the terminology is borrowed from neurology, the analogy between an NN and the brain can well be ignored. The idea behind the computation in an NN is that inputs from the external world activate a subset of the neurons (the *input neurons*), which, in turn, elaborate the input and transmit the results of this local computation to other neurons until some subset of neurons (the *output neurons*) provide the final results to the external world. All neurons that are neither input nor output are *internal*. In Figure 7.1 an example of a generic NN is given.

Clearly the above definition covers an unlimited number of structures. Moreover, the types of function computed by the elementary units increase the variety of NNs along another dimension. If no limits are set on the network's structure and local computation, it is almost impossible to analyze its behavior. Thus precisely defined network types have been introduced and studied. For this, a model of the neuron is needed. A very basic model of a neuron ν_j, represented in Figure 7.2, includes the following elements.

Formal model of a neuron

- *A set of input links (synapses)*, which transmit to a neuron ν_j the signals $(x_1, \ldots, x_r)$ arriving from r other neurons or from the external world. Each link $x_{i,j}$ $(1 \leqslant i \leqslant r)$ has an associated weight $w_{i,j}$, which multiplies the input signal x_i. If $w_{i,j} > 0$ the link is *excitatory*, whereas it is *inhibitory* if $w_{i,j} < 0$. If $w_{i,j} = 0$, the link is not present in the network. An output link y_j carries the *activation value* of the neuron.

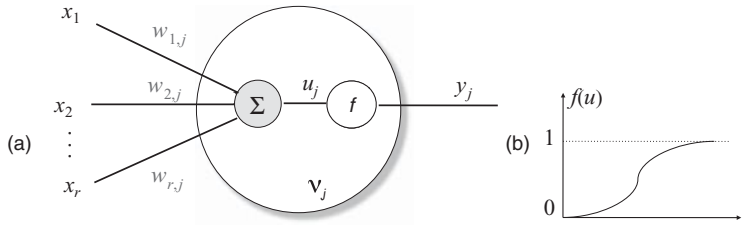


Figure 7.2 A basic model of a neuron in an artificial neuron network. Within the neuron, (a) an adder computes u_j as a weighted sum $\sum_{i=1}^{r} w_{i,j}x_i$ of the input signals and (b) the output is then calculated by a limiting activation function $f(u)$ (in the figure it is logistic).

- *An adder*, which sums up the input signals weighted by the respective $w_{i,j}$ values. Thus, the adder computes the sum

$$u_j = \sum_{i=1}^{r} w_{i,j}x_i.$$

  The quantity u_j is known as the *post-synaptic potential*.

- *An activation function* $y_j = f(\cdot)$, which limits the amplitude of the output signal. Typically, the output range of a neuron is the interval $[0, 1]$ or $[-1, +1]$. The output y_j is then computed from

$$y_j = f(u_j - \tau_j),$$

  where τ_j is the threshold typical of neuron ν_j. The activation function may take a wide variety of formats; two common ones are the *threshold function*,

$$f(u) = \begin{cases} 1 & \text{if } v \geqslant 0, \\ 0 & \text{if } v < 0, \end{cases} \tag{7.1}$$

  and the *logistic function*,

$$f(u) = \frac{1}{1 + e^{-au}}. \tag{7.2}$$

  The numerical value a in (7.2) is a *slope parameter*.

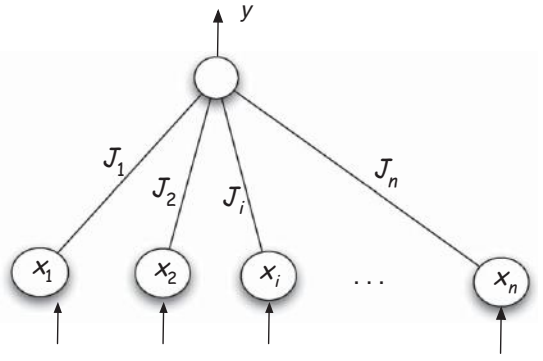


Figure 7.3 The perceptron. All the neurons in the input layer are connected to the output neuron, with links labeled J_i $(1 \leqslant i \leqslant n)$. The J_i represent the quenched disorder.

The perceptron

The first studies on NNs date back to the 1950s, when Rosenblatt (1958) proposed a very simple network model, which nevertheless aroused considerable interest: the *perceptron*. Its structure is simply organized into two *layers*, one of input neurons (the *input layer*) and the other constituted by a single output neuron (the *output layer*), as represented in Figure 7.3.[1] The input layer receives the signals $(x_1, \ldots, x_n)$ from a "retina"; the output neuron elaborates these signals and provides the result y. Since there is a unique output node, the weights can simply be defined as J_i $(1 \leqslant i \leqslant n)$. They represent the quenched disorder introduced in Chapter 2.

Central to the operation of Rosenblatt's perceptron is the *McCulloch–Pitts model* of the neuron, depicted in Figure 7.2. Given an external threshold τ, the adder produces the value

$$u = \sum_{i=1}^{n} w_i x_i - \tau. \tag{7.3}$$

The activation function y_j is a *hard limiter*, which outputs 1 if $u \geqslant 0$ and 0 otherwise. The purpose of the perceptron is to distinguish between two classes of objects, c_1 and c_2. The decision boundary is the hyperplane

$$u = \sum_{i=1}^{n} w_i x_i - \tau = 0. \tag{7.4}$$

[1] Nevertheless, this basic construct is regarded as a *single-layer perceptron*; the output layer is disregarded in this terminology.

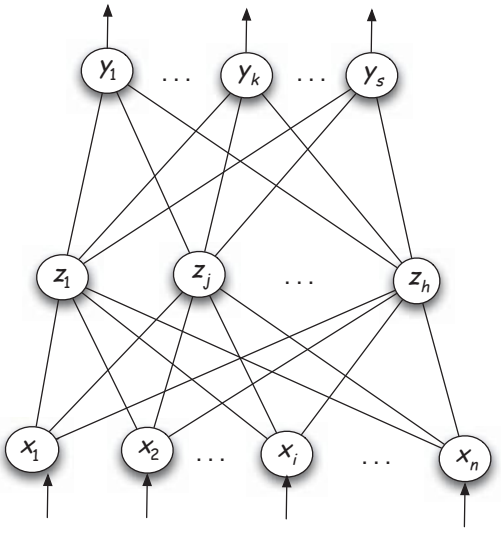


Figure 7.4 Multi-layer perceptron with a single layer of hidden units.

If $u \geqslant 0$ then the input is assigned to class c_1, otherwise to class c_2. The interest in the perceptron was mainly due to the possibility of training it to recognize inputs and classify them. To this end, a set of labeled training examples is shown to the perceptron, and the weights are updated each time a new example (or a set of examples) is seen. Rosenblatt proved that the updating algorithm converges and correctly separates the two classes by a hyperplane, if the classes are indeed linearly separable. Notwithstanding Rosenblatt's optimism about the potential of the perceptron, the use of a step function for $f(u)$ did not allow effective training. A critical analysis of the perceptron, published by Minsky and Papert (1969), pointed out a number of essential weaknesses, as a consequence of which research on neural networks almost died out.

Multi-layer perceptron

Interest in neural networks returned following the proposal, by Rumelhart, Hinton, and Williams (Rumelhart *et al.*, 1986; Rumelhart and McClelland, 1986), of a new network model, the *multi-layer perceptron* (MLP), with non-linear but differentiable functions $f(u)$. The new model overcame the problem of the original perceptron's step function, and was also provided with an effective training method, the *error back-propagation* algorithm. The MLP, whose scheme is illustrated in Figure 7.4, is a layered, directed, network, in which neurons are organized into an *input layer*, an *output layer*, and one or more *hidden layers*. The neurons in the internal layers are called *hidden units*. Only neurons in adjacent layers can be connected. The MLP is a *feed-forward* network, where

the information flows from input to output. It can potentially realize any kind of nonlinear separation between classes.

In contrast, even though the simplicity of the perceptron makes it unrealistic for practical application (it discriminates only between linearly separable classes of objects), the same simplicity allows for detailed studies of its behaviors. In particular, early research on the emergence, in learning, of phenomena typical of statistical physics concentrated on neural networks and, in particular, on the perceptron (Kinzel, 1998; Engel and Van den Broeck, 2001; Biehl *et al.*, 2000; Nishimori, 2001).

7.1.1 The perceptron

In order to describe the results obtained by the statistical mechanics approach to learning in a perceptron, we need to introduce some notions first. Let n be the number of input units in the network. Given n and the activation function $f(\cdot)$, a given perceptron $\mathbf{P}$ differs from any other only with respect to the synaptic weights on the links between the input and the output neurons. The net has n degrees of freedom. Let $\vec{\mathbf{x}} = (x_1, \ldots, x_n)$ be the input vector with n components, $\vec{\mathbf{W}} = (w_1, \ldots, w_n)$ the corresponding vector of weights, and y the output. The vector $\vec{\mathbf{x}}$ belongs to the input space $\mathcal{X}$. Let us denote by $\mathcal{W}$ the set of all possible weight assignments. Then, any network can be identified by its corresponding weight vector $\vec{\mathbf{W}}$.

The network $\vec{\mathbf{W}}$, called the "student", must learn a decision boundary between two classes c_1 (corresponding to an output $y = +1$) and c_2 (corresponding to an output $y = -1$) by looking at a subset of $\mathcal{X}$, namely the learning set $\mathcal{S}_L = \{\vec{\mathbf{x}}^{(i)} | 1 \leqslant i \leqslant r\}$, containing r examples. Let $\vec{\mathbf{T}}$ be the "teacher", namely a network with weights that allow perfect discrimination between the two classes, and let the output be y_T. The goal of learning is to exploit $\mathcal{S}_L$ to let the network $\vec{\mathbf{W}}$ become as close as possible to $\vec{\mathbf{T}}$. In order to measure the progress of $\vec{\mathbf{W}}$, a performance measure is needed; a suitable one is the following:

The student network

The teacher network

$$d(\vec{\mathbf{W}}, \vec{\mathbf{T}}, \vec{\mathbf{x}}) = \mathbf{1}(-y(\vec{\mathbf{x}}) \cdot y_T), \tag{7.5}$$

where $\vec{\mathbf{x}}$ is a given input vector, and $1(u)$ is the *step function*, which assumes the value 1 if $u \geqslant 0$ and 0 otherwise. Clearly, $d = 1$ iff $\vec{\mathbf{W}}$ and $\vec{\mathbf{T}}$ classify the input vector $\vec{\mathbf{x}}$ differently (an error has occurred). Using (7.5) the training error, i.e., the error that results when $\vec{\mathbf{W}}$ performs on the learning set $\mathcal{S}_L$, can be computed as follows:

$$\varepsilon_t = \frac{1}{r} \sum_{i=1}^{r} d\left(\vec{\mathbf{W}}, \vec{\mathbf{T}}, \vec{\mathbf{x}}^{(i)}\right). \tag{7.6}$$

As discussed in Chapter 5, choosing a student with a training error as low as possible might not be a good idea, as there is a danger of overfitting. We need to

Generalization error

introduce the *generalization error* $\varepsilon(\vec{\mathbf{W}}, \vec{\mathbf{T}})$, which is defined as the average of d over all possible inputs $\vec{\mathbf{x}}$:

$$\varepsilon = \varepsilon(\vec{\mathbf{W}}, \vec{\mathbf{T}}) = \sum_{\vec{\mathbf{x}} \in \mathcal{X}} \mathtt{P}(\vec{\mathbf{x}})\, d(\vec{\mathbf{W}}, \vec{\mathbf{T}}, \vec{\mathbf{x}}). \tag{7.7}$$

The average (7.7) is performed over the quenched disorders (the training examples). The probability distribution $\mathtt{P}(\vec{\mathbf{x}})$ is usually fixed but unknown. The error ε can also be interpreted as the probability that $\vec{\mathbf{W}}$ is in error on an input vector $\vec{\mathbf{x}}$ *randomly drawn* from $\mathtt{P}(\vec{\mathbf{x}})$. What we want to obtain is an error ε as low as possible.

The behavior of an NN in the limit of large n can be rewritten in terms of statistical physics quantities. More precisely, the weights $\vec{\mathbf{W}}$ are the *couplings* $\vec{\mathbf{J}}$ of the net (hence, $w_i = J_i$), and they can also be considered as describing *quenched disorder*. Actually, the generalization error ε given in (7.7) is still a function of $\vec{\mathbf{J}}$. In order to analyze the behavior of the perceptron quantitatively, we have to make some assumptions about the range of values assumed by the quantities involved. If the output y can only assume values ± 1 then the perceptron is called *binary*, and $f(u)$ is a threshold function. If, on the contrary, $f(u) = u$, the perceptron becomes *linear*. In an analogous way, the vector $\vec{\mathbf{W}} = \vec{\mathbf{J}}$ may have continuous or binary components; in the latter case, the perceptron is called an *Ising perceptron*. For example, an Ising binary perceptron has both the J_i $(1 \leqslant i \leqslant n)$ and y equal to ± 1. If $\vec{\mathbf{J}}$ has continuous components, a special case is that in which all possible $\vec{\mathbf{J}}$ vectors have the same length, $\sqrt{n}$; this type of perceptron is said to be *spherical*.

In the following we will consider the case where

$$\vec{\mathbf{W}}^2 = \vec{\mathbf{J}}^2 = \sum_{i=1}^{n} J_i^2 = n. \tag{7.8}$$

Moreover, let

$$\vec{\mathbf{x}}^2 = \sum_{i=1}^{n} x_i^2 = n. \tag{7.9}$$

Hamiltonian of the perceptron

By using the definition of energy introduced in Chapter 1, the Hamiltonian of the perceptron can be written as

$$H(\vec{\mathbf{x}}) = -\sum_{i=1}^{n} J_i x_i y. \tag{7.10}$$

If the perceptron's behavior is governed by a threshold function $f(u)$, it can be shown that on the one hand the Hamiltonian is upper-bounded. In fact, from

(7.3) we have that $H(\vec{\mathbf{x}}) = -uf(u)$, with $\tau = 0$. If $u \geqslant 0$ then $f(u) = 1$ and $H = -u < 0$; if $u < 0$ then $f(u) = 0$ and $H = 0$. On the other hand, the Hamiltonian is also lower-bounded, because $|u| \leqslant n$. As a consequence the perceptron will reach an *attractor* state, where it will stay (Engel and Van den Broeck, 2001). Since in the thermodynamic limit the number of degrees of freedom, n, of the perceptron diverges, we may expect that for learning to take place the number of examples shown to the perceptron must diverge too. As it turns out, $r = |\mathcal{S}_L|$ diverges but the ratio $\alpha = r/n$ must remain constant.

Requiring that $\vec{\mathbf{J}}, \vec{\mathbf{T}}$, and $\vec{\mathbf{x}}$ have length $\sqrt{n}$ does not affect the performance of the perceptron, because only the angle between $\vec{\mathbf{J}}$ and $\vec{\mathbf{x}}$ is important, as we now explain. The boundary between the two classes provided by equation (7.4) can be written as

$$\sum_{i=1}^{n} J_i x_i = \vec{\mathbf{J}} \cdot \vec{\mathbf{x}} = |\vec{\mathbf{J}}||\vec{\mathbf{x}}| \cos\theta = 0, \tag{7.11}$$

where "$\cdot$" denotes the scalar product of two vectors. As neither $|\vec{\mathbf{J}}|$ nor $|\vec{\mathbf{x}}|$ can become zero, the only relevant quantity is $\cos\theta$, i.e., the cosine of the angle between the two vectors.

As already mentioned, the goal of learning is to reduce the distance between the student $\vec{\mathbf{J}}$ and the teacher $\vec{\mathbf{T}}$. In order to evaluate their distance, the *overlap* R is introduced:

Overlap between networks

$$R = \frac{\vec{\mathbf{W}} \cdot \vec{\mathbf{T}}}{n} = \frac{\vec{\mathbf{J}} \cdot \vec{\mathbf{T}}}{n} = \frac{1}{n} \sum_{i=1}^{n} w_i T_i. \tag{7.12}$$

The overlap R is 0 when $\vec{\mathbf{J}}$ and $\vec{\mathbf{T}}$ are orthogonal vectors, whereas it is a maximum when they coincide. In fact, the *generalization* error ε of the network can be written as:

$$\varepsilon = \frac{1}{\pi} \arccos R. \tag{7.13}$$

In Figure 7.5, a circle of unit radius illustrates the relation between $\vec{\mathbf{J}}$ and $\vec{\mathbf{T}}$. As $R = \cos\theta$, the error is equal to the ratio of the total length of the two arcs defining the hatched regions ($2 \arccos R$) and the whole circumference (2π).

In order to set the weights $\vec{\mathbf{J}}$, a simple learning scheme, the *Gibbs rule*, can be exploited. Let us consider exactly the set of all couplings $\vec{\mathbf{J}}$ that classify the training examples seen so far as the teacher. These couplings are said to be *compatible* and they constitute the *version space*. We want to compute the generalization error of a vector $\vec{\mathbf{J}}$ drawn randomly from the version space. When a new example is shown to the network, the version space can only shrink and the generalization error decrease; the reason is that some more incompatible couplings are rejected, and the generalization error is given by (7.7), in which the

Version space

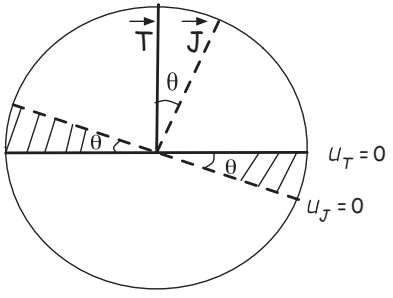


Figure 7.5 Relationship between the student $\vec{\mathbf{J}}$ and the teacher $\vec{\mathbf{T}}$. The hyperplanes orthogonal to the vectors represent the decision boundaries between the classes. Above the hyperplane $u_T = 0$ all the objects belongs to class c_1, whereas below the hyperplane all the objects belong to class c_2; as $\vec{\mathbf{T}}$ is the teacher this is the correct classification. However, the student puts in class c_1 objects that are above the hyperplane $u_J = 0$ and in class c_2 those that are below it. Thus all objects located in the hatched regions are misclassified by the student.

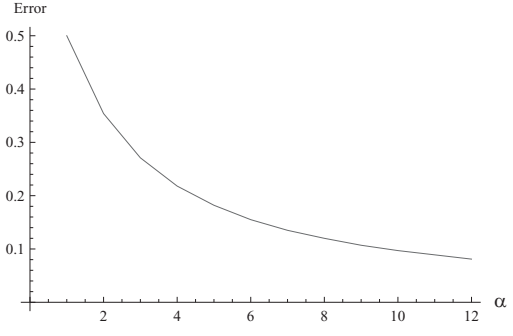


Figure 7.6 Computation of the generalization error $\varepsilon_\infty(\alpha)$ as a function of α, from (7.14).

proportion of compatible couplings now increases. The obtained generalization error characterizes the *typical* performance of a compatible student. It is possible to show (Engel and Van den Broeck, 2001) that, in the thermodynamic limit, the generalization error is the following function of α:

The parameter $\alpha = r/n$ is the ratio of the number of training examples and the degree of freedom of the perceptron.

$$\varepsilon_\infty(\alpha) = \operatorname{argmax}\left[\frac{1}{2}\ln\sin^2(\pi\varepsilon) + \alpha\ln(1-\varepsilon)\right]. \tag{7.14}$$

For $\alpha = 0$ we have $\varepsilon = 0.5$, i.e., the student is orthogonal to the teacher and the learned classifier performs random guesses. For $\alpha \to \infty$ we expect $\varepsilon \to 0$.

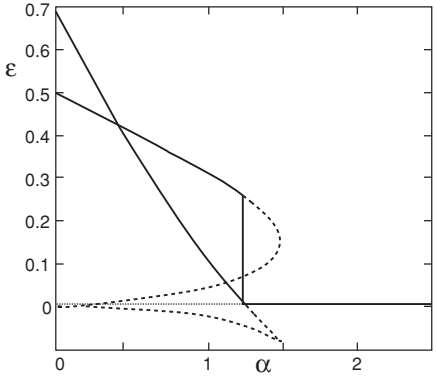


Figure 7.7 Phase transition of the generalization error ε as a function of α. The critical value of the control parameter is $\alpha_{cr} \cong 1.245$. The solid curve starting at 0.5 represents the generalization error and the solid curve starting at 0.7 represents the entropy. The broken lines correspond to metastable states. Reprinted from Engel and Van den Broeck (2001).

The computation of $\varepsilon_\infty(\alpha)$ is described in Figure 7.6. For large α, equation (7.7) gives:

$$\varepsilon \sim \frac{1}{\alpha}. \tag{7.15}$$

The above simple analysis yields qualitatively correct results. In order to obtain quantitatively correct results, methods from statistical physics must be used, in particular the *annealed approximation* and *Gardner's analysis* (Engel and Van den Broeck, 2001). The most difficult part of the computation is handling the quenched disorder, namely, the training example distribution. The obvious quantity to use to compute the generalization error would seem to be the volume of version space but unfortunately this is not a self-averaging quantity (see Section 2.6.1). In fact the correct quantity for use in generalization problems is the *entropy*, which is the logarithm of the version space volume; its average value can be computed in the thermodynamic limit, $n \to \infty$, using the replica trick with the symmetry *ansatz* (see Section 2.7). The results are given in Figure 7.7.

In Figure 7.6 the generalization error shows a smooth decrease from $\varepsilon = 0.5$ to the asymptotic limit $\varepsilon = 0$. If some network parameters are constrained to be Boolean, new phenomena emerge. For example, if an Ising perceptron with Boolean $\vec{\mathbf{J}}$ and $\vec{\mathbf{x}}$ that is learning from another Ising perceptron $\vec{\mathbf{T}}$ is considered, then statistical physics methods reveal the presence of a first-order phase transition in the generalization error ε (the order parameter) with respect to the control parameter α. More precisely, the generalization error drops to 0 at the value $\alpha_{cr} \cong 1.245$, showing an abrupt transition to perfect learning. The meaning of

Transition to perfect generalization

the phase transition is that if an Ising perceptron classifies $1.245\,n$ random inputs in the same way as the teacher, it will classify correctly all 2^n possible inputs. This results was obtained by applying the replica symmetry method (Engel and Van den Broeck, 2001).

These results confirmed that obtained by Seung *et al.* (1992), who studied perceptron learning with equilibrium statistical mechanics methods. The exact quenched disorder analysis was approximated in two cases, the high-temperature limit and the annealed approximation. The authors assumed *realizable* target rules, i.e., perfect decision boundaries between two classes, provided by a teacher perceptron with the same structure as the learner's. They studied four types of perceptron determined by the continuous or discrete nature of the weights and of the output. If the weights are continuous, the generalization error varies with continuity and shows the typical inverse-power-law behavior for increasing number of training examples. When the weights are discrete, a first-order phase transition appears: if the output is linear, the transition is from a state of poor generalization to one of good generalization followed by an inverse power law decay toward zero. If the output is discrete as well then the transition is to perfect generalization. Monte Carlo simulations demonstrate that the results are quantitatively correct at high temperatures and qualitatively correct at low temperatures.

Continuous weights do not generate discontinuities.

A comprehensive account of phase transition appearances in neural networks was provided by Watkin et al. (1993), who proposed a general setting for investigation of the issue and reported results for several types of networks, including multi-layered ones.

7.1.2 Multi-layer perceptrons

Multi-layer perceptrons were studied by Biehl *et al*. (2000). The authors considered the two-layer perceptron shown in Figure 7.8, which has also been called the *soft-committee machine* (see also Kinzel, 1998).

Each hidden unit z_j $(1 \leqslant j \leqslant K)$ is connected to all input units x_i $(1 \leqslant i \leqslant n)$, with weights J_{ij} $(1 \leqslant i \leqslant n, 1 \leqslant j \leqslant K)$. By defining as $\vec{\mathbf{J}}^{(j)}$ $(1 \leqslant j \leqslant K)$ a vector whose components are the weights J_{ij}, the *activation value* of the hidden units can be given as

$$z_j = \frac{1}{\sqrt{n}}\, \vec{\mathbf{J}}^{(j)} \cdot \vec{\mathbf{x}} \qquad (1 \leqslant j \leqslant K),$$

and the output as

$$y(\vec{\mathbf{x}}) = \frac{1}{\sqrt{K}} \sum_{j=1}^{K} \operatorname{erf}\left(\frac{z_j}{\sqrt{2}}\right).$$

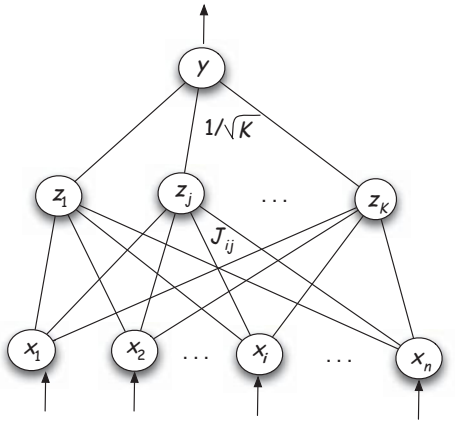


Figure 7.8 Two-layer perceptron studied by Biehl *et al.* (2000). The input layer has n neurons, and there are K hidden units and one output unit. The weights of the connection between the hidden and output neurons are all equal to $1/\sqrt{K}$.

Learning occurs, as usual, through the provision of a training set $\mathcal{S}_L = \{\vec{\mathbf{x}}^{(k)} | 1 \leqslant k \leqslant m\}$ containing m labeled examples. The exact rule that the student network is supposed to learn corresponds to an identical two-layer perceptron, with weights $\vec{\mathbf{B}}^{(j)}$ between the input and the hidden units and output $v(\vec{\mathbf{x}})$. Both $y(\vec{\mathbf{x}})$ and $v(\vec{\mathbf{x}})$ being continuous, the empirical error is evaluated as the squared difference between the true and the learned outputs for the examples belonging to $\mathcal{S}_L$:

$$\varepsilon_t = \frac{1}{m} \sum_{k=1}^{m} \left[y(\vec{\mathbf{x}}^{(k)}) - v(\vec{\mathbf{x}}^{(k)}) \right]^2 . \tag{7.16}$$

As already mentioned the examples in $\mathcal{S}_L$, extracted from an unknown but fixed probability distribution $\mathbf{D}$, act as a *quenched disorder*. Thus, the generalization error ε can be expressed as follows:

$$\varepsilon = \overline{[y(\vec{\mathbf{x}}) - v(\vec{\mathbf{x}})]^2}, \tag{7.17}$$

where the overbar denotes the average with respect to the distribution $\mathbf{D}$ of the quenched disorder. The inputs x_i are independent identically distributed Gaussian variables with zero mean and unit variance. From a statistical physics perspective the product $m\varepsilon_t$ can be considered as the extensive *energy* of the system, which can be used, as in expression (2.12), to assign the probabilities $\exp[-(\beta m \varepsilon_t)]$ to an ensemble of networks (Kinzel, 1998) each satisfying the condition $(\vec{\mathbf{J}}^{(j)})^2 = n$.

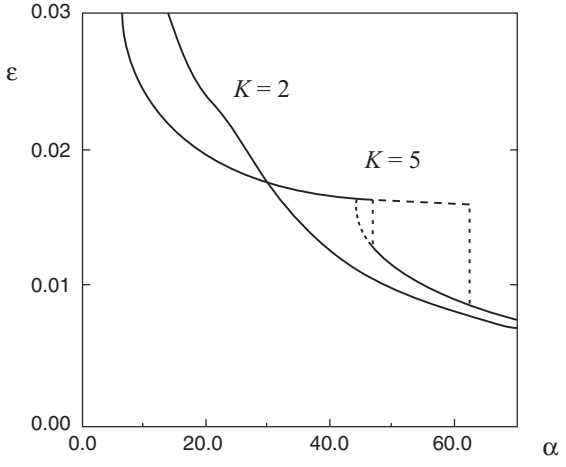


Figure 7.9 Learning curves for small K (reprinted from Biehl *et al*., 2000). For $K = 2$ a second-order phase transition occurs at the critical value $\alpha_{cr}(2) \simeq 23.7$, whereas for $K = 5$ a first-order transition occurs at $\alpha_{cr}(5) \simeq 46.6$.

To develop the analysis further, Biehl *et al*. (2000) considered the case of high temperatures, i.e., $\beta \to 0$. Under the above conditions the generalization error shows, as in the case of the perceptron described in Section 7.1.1, a phase transition for a critical value of the number of training examples. More precisely, the control parameter is $\alpha = \beta m/(nK)$. The nature of the transition and the critical values depend on K. In particular, for $K = 2$ the transition is second-order and occurs at $\alpha_{cr}(2) \simeq 23.7$, whereas the transition is first-order for $K \geqslant 3$. The main results are given in Figure 7.9.

7.2 Propositional learning approaches

Statistical physics methods have been applied, in the past, not only to analyze the behavior of neural networks but also in more general settings, including the learning of Boolean functions, learning with support vector machines, and symbolic learning.

7.2.1 Learning Boolean functions

Haussler *et al*. (1994) provided rigorous learning curve bounds using methods derived from statistical physics. More precisely, they showed that the bounds

are often tighter and more realistic than those provided by the classical Vapnik–Chervonenkis theory. The learning curves may show discontinuous behaviors, such as the emergence of phase transitions, as well as power law asymptotic behaviors that are not explained within that theory.

The Vapnik–Chervonenkis theory of learning a function in a class $\mathcal{F}$, via minimization of the empirical error on a random sample of m examples, leads to bounds on the generalization error which decrease at a rate $\mathcal{O}(d/m)$, where d is the *Vapnik–Chervonenkis dimension* of the class, if the target function is in $\mathcal{F}$. If the target function is not in $\mathcal{F}$, the generalization error decreases at the much lower rate of $\mathcal{O}(d/\sqrt{m})$. These bounds are general, in the sense that they are valid for any class $\mathcal{F}$, any input distribution, and any target function. The only problem-specific quantity is d, which is typical of $\mathcal{F}$. In addition, these bounds are the best possible distribution-independent bounds. Nonetheless, various workers have pointed out discrepancies between the predictions of the theory and the experimental results. Statistical physics methods have helped to show where the Vapnik–Chervonenkis bounds fail to model the true behavior of learning curves, by revealing unexpected behaviors such as the emergence of phase transitions or asymptotic power laws whose exponent is neither 1 nor $1/2$.

Generalization error bound

In Haussler *et al.* (1994), the authors exploited the annealed approximation and the thermodynamic limit to derive a precise and rigorous theory of learning curves. The thermodynamic-limit setting allows the notion of a correct *scale* to be introduced for describing the learning curves. These last derive from a competition between an *entropy* function and an *energy* function. The entropy is measured by the logarithm of the number of hypotheses as a function of the generalization error ε. The energy is measured by the probability of minimizing the empirical error on a random sample as a function of the generalization error ε.

Haussler and co-workers started their analysis from a very simple case, in which a Boolean function over an input space $\mathcal{X}$ is learned using a training set $\mathcal{S}_L$ of m labeled examples. The target function $f(\vec{\mathbf{x}})$ is known to belong to a class $\mathcal{F}$ containing a finite number of $\{0,1\}$-valued functions. Moreover, there is a probability distribution $\mathbf{D}(\vec{\mathbf{x}})$ defined over $\mathcal{X}$. If $h(\vec{\mathbf{x}})$ is a Boolean function output of some learning algorithm, the generalization error can be written as

$$\varepsilon(h) = \mathtt{P}_{\vec{\mathbf{x}}\sim D}[h(\vec{\mathbf{x}}) \neq f(\vec{\mathbf{x}})]. \tag{7.18}$$

In order to perform their analysis of the generalization error, the version space (see Section 7.1.1) is used. Formally, the version space is defined by

Version space

$$\mathcal{VS}(\mathcal{S}_L) = \{h \in \mathcal{F} \mid \forall\, [\vec{\mathbf{x}}, f(\vec{\mathbf{x}})] \in \mathcal{S}_L, h(\vec{\mathbf{x}}) = f(\vec{\mathbf{x}})\}. \tag{7.19}$$

In other words, it contains all the hypotheses (functions) that are *consistent* with the training set. Then, an η-ball $B(\eta)$ is defined around the target function:

$$B(\eta) = \{h \in \mathcal{F} \mid \varepsilon(h) \leqslant \eta\}. \tag{7.20}$$

Using a method close to the annealed approximation, the following *union bound* can be established:

$$\mathrm{P}_{\mathcal{S}_L}\left[\mathcal{VS}(\mathcal{S}_L) \subseteq B(\eta) \geqslant 1 - \delta\right], \tag{7.21}$$

where

$$\delta = \sum_{h \in \overline{B(\eta)}} \left[1 - \varepsilon(h)\right]^m .$$

The main results can be restated as follows.

Theorem 7.1 {*Haussler* et al.*, 1994*} *Let $\mathcal{F}$ be any finite class of Boolean functions. For any $0 < \delta \leqslant 1$, with probability at least $1 - \delta$, any function $h \in \mathcal{F}$ that is consistent with m random examples of the target function satisfies the relation $\varepsilon(h) \leqslant \eta$, where η is the smallest value satisfying the condition*

$$\sum_{h \in \overline{B(\eta)}} \left[1 - \varepsilon(h)\right]^m \leqslant \delta.$$

Haussler *et al*. derived also a standard bound, which they call the *cardinality bound*,

$$\varepsilon(h) \leqslant \frac{1}{m} \ln\left(\frac{|\mathcal{F}|}{\delta}\right). \tag{7.22}$$

As acknowledged by the authors themselves, the results they provide have more theoretical value than practical value. The greater tightness of the bounds compared with those provided by the Vapnik–Chervonenkis theory comes, in fact, at the expense of a greater amount of information required, namely, knowledge of the example distribution, as against knowledge of the class $\mathcal{F}$ only in the Vapnik–Chervonenkis theory. In addition, in the method of Haussler *et al*. the analysis is limited to function classes of finite cardinality.

7.2.2 Support vector machines

Support vector machines (SVMs) can be considered as a generalization of single-layer perceptrons (see Section 7.1); they project the initial examples, belonging to a set of non-linearly separable classes, onto a high-dimensional feature space, where linear classifiers can be found. Support vector machines were introduced by Cortes and Vapnik (1995), and have proved to be effective learners for a variety of tasks.

Analogously to single and multi-layer perceptrons, SVMs too lend themselves to a statistical mechanics analysis, which provides insights into the typical behavior of the generalization error (Dietrich *et al.*, 1999; Malzahn and Opper, 2005). Let n be the dimension of the input vectors (the examples) to be classified, N the number of features in the transformed space, and m the number of training examples. Let us define, as for the perceptron, $\alpha = m/n$. Dietrich *et al.* (1999) took the thermodynamic limit $n \to \infty$ for various scales of increase in m. As the number of learned parameters is N, one may expect that the decrease to zero in the generalization error ε should occur for $m = \mathcal{O}(N)$. On the contrary, however, ε becomes small even for $m = \mathcal{O}(n)$. Thus a reasonable *ansatz* is $m = \alpha n^{\ell}$. If both the student and the teacher are expressed via quadratic kernels, these authors found that for $\ell = 1$, i.e., $m = \alpha n$, the student is able to learn only the linear part of the teacher's rule; then, for increasing α, the generalization error reaches a plateau where

$$\varepsilon(\alpha) - \varepsilon(\infty) \sim \alpha^{-1}.$$

If the number of examples increases on the larger scale $m = \alpha n^2$, the well-known asymptotic behavior $\varepsilon \sim \alpha^{-1}$ is found. If the kernels are polynomials of degree p, a sequence of plateaus appears.

A further result obtained by Dietrich *et al.* (1999) was that SVMs show a resistance to overfitting. In fact, by letting a quadratic SVM learn a linear rule, successful learning occurs on the scale $m = \alpha n$ even though the generalization error decay is $\varepsilon \sim \alpha^{-2/3}$, somewhat slower than the classical α^{-1}.

Support vector machines have also been investigated by Malzahn and Opper (2005), whose main goal was to prove that methods from statistical physics can be easily extended to become applicable to real learning algorithms working on real data. To achieve their goal, the authors used a combination of the replica approach and the variational method.

More recently, an extension of SVMs, namely *multiple instance support vector machines* (MI-SVMs) were investigated by Gaudel *et al.* (2008). The MI-SVMs combine multiple instance learning (Dietterich *et al.*, 1997) with SVMs and represent an intermediate step between propositional and fully relational learning. In the MI setting, each example $\vec{\mathbf{z}}^{(i)}$ is a "bag" of n_i propositional instances $x_1^{(i)} \ldots, x_{n_i}^{(i)}$, where $\vec{\mathbf{z}}^{(i)}$ is positive iff some of its instances satisfy the (propositional) target concept. For MI problems the kernel K of two bags of instances is defined as the average of the kernels k of pairs of instances:

$$K(\vec{\mathbf{z}}^{(i)}, \vec{\mathbf{z}}^{(j)}) = \frac{1}{f_n(\vec{\mathbf{z}}^{(i)}) f_n(\vec{\mathbf{z}}^{(j)})} \sum_{k=1}^{n_i} \sum_{h=1}^{n_j} k(x_k^{(i)}, x_h^{(j)}). \tag{7.23}$$

In (7.23) $f_n(\vec{\mathbf{z}})$ is a normalization function. The approach is efficient under the so-called *linearity* assumption, i.e., an example is positive iff it contains (at least) one instance satisfying the target concept.

In order to investigate the quality of the propositionalization induced by relational kernels, the authors generated a set of artificial problems, each consisting of a learning set $\mathcal{S}_L = \{(\vec{\mathbf{z}}^{(1)}, y_1), \ldots, (\vec{\mathbf{z}}^{(m)}, y_m)\}$ and a test set $\mathcal{S}_T = \{(\vec{\mathbf{z}}'^{(1)}, y'_1), \ldots, (\vec{\mathbf{z}}'^{(m')}, y'_{m'})\}$. The training set $\mathcal{S}_L$ induces a propositionalization of the domain space, mapping every MI example $\vec{\mathbf{z}}$ to the m-dimensional real vector $\Phi_{\mathcal{L}}(\vec{\mathbf{z}}) = (K(\vec{\mathbf{z}}^{(1)}, \vec{\mathbf{z}}), \ldots, K(\vec{\mathbf{z}}^{(m)}, \vec{\mathbf{z}}))$. Let $\mathcal{R}_{\mathcal{L}}$ describe this new representation. The novelty of the proposed methodology is that it rewrites an MI-SVM learning problem $(\mathcal{S}_L, \mathcal{S}_T)$ as a constraint satisfaction problem $Q(\mathcal{S}_L, \mathcal{S}_T)$ in $\mathcal{R}_{\mathcal{L}}$. If ε denotes the generalization error of the optimal linear classifier $h^*_{\mathcal{L}}$ defined on $\mathcal{R}_{\mathcal{L}}$ then the authors lower-bounded ε's expectation as follows:

$$\mathbb{E}_m[\varepsilon] \geqslant 1 - (\hat{\tau}_{m,m'} + \eta)^{1/m'}, \tag{7.24}$$

where the average is computed over a set of learning problems $\{(\mathcal{S}_L^{(r)}, \mathcal{S}_T^{(r)}) | 1 \leqslant r \leqslant R\}$, where $\eta > 0$ is any number and $\hat{\tau}_{m,m'}$ is the fraction of CSPs $Q(\mathcal{S}_L^{(r)}, \mathcal{S}_T^{(r)})$ that are satisfiable.

For the experiments, 40 learning problems were generated for each assignment of values to a set of 12 control parameters, by choosing independently the target concept c, the training set $\mathcal{S}_L$, and the test set $\mathcal{S}_T$. The kernels were Gaussian and the normalization function $f(\vec{\mathbf{z}})$ was set to the number of instances (P_{ic}, N_{ic}) in the example $\vec{\mathbf{z}}$. Looking at the influence of the various parameters involved, it was noticed that there is a region where all the CSPs $Q(\mathcal{S}_L^{(r)}, \mathcal{S}_T^{(r)})$ are unsatisfiable; for instance, in the plane (P_{ic}, N_{ic}) this region appears along the diagonal $P_{ic} = N_{ic}$, as expected. The size of this region decreases as the number of training examples increases (left-hand column in Figure 10.1) but increases as the number of test examples increases (right-hand column in Figure 10.1). Even though not directly related to the appearance of a phase transition, the reported results reveal the emergence of discontinuous phenomena in this type of propositionalization.

7.2.3 Decision tree induction

Work similar in spirit to that described in the previous section was carried out by Baskiotis and Sebag (2004). Taking inspiration from the phase transition paradigm, their goal was to attach a principled *competence map* to a learning algorithm, characterizing the regions (if any) of the problem space where the algorithm *typically* succeeds or fails. Such competence maps could then be exploited as look-up tables for the choice of a learning algorithm. The proposed approach was applied, as a particular case, to the decision tree learner C4.5 (Quinlan, 1993). The target concept space was set to formulas in *disjunctive*

Competence map

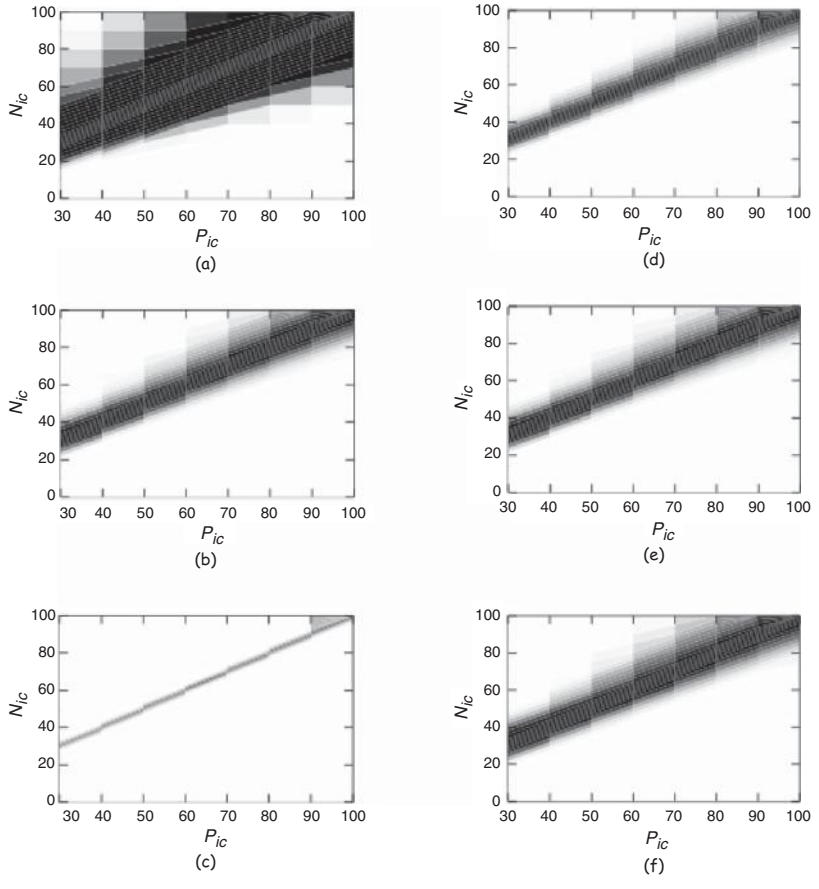


Figure 7.10 Fraction of satisfiable CSPs $Q(\mathcal{S}_L, \mathcal{S}_T)$ in the plane (P_{ic}, N_{ic}) out of 40 runs. Reprinted from Gaudel *et al*. (2008) with permission.

normal form and, hence, the Rule learning mode of C4.5 was used. In this mode a set of decision trees is constructed, pruned, and compiled into rules; the rules are then filtered and ordered on the training set; finally, the rule set is used as a decision list on the test examples.

For the experiments, four control parameters were considered: the number n of Boolean variables representing the problem domain; the number k of (distinct) terms γ_i in the target concept, each term being the conjunction of a set of (distinct) literals y_i, each either a variable (x_i) or its negation $(\bar{x}_i)$; the number ℓ of literals in a term; the imbalance ratio r, which is the fraction of positive learning instances. Assuming that all terms have the same length, the target concept space is actually a restriction of the DNF language, termed (k, ℓ)-term DNF.

The assumption of uniform ℓ was later on relaxed in the paper.

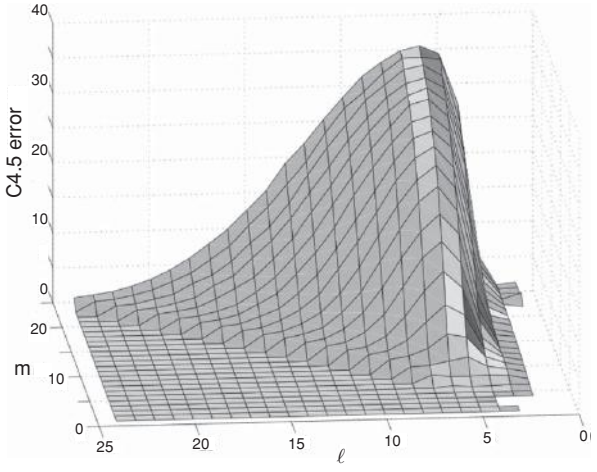


Figure 7.11 Competence map for the C4.5 percentage error for (k, ℓ)-term DNF, represented in the plane (m, ℓ) with $k = 15$ and $r = 1/2$. Reprinted from Gaudel *et al*. (2008) with permission.

For each setting (n, k, ℓ, r), 100 learning problems $\mathcal{L}_i(n, k, \ell, r)$ were generated. Each learning problem consisted of a target concept, a training set, and a test set of examples (globally, 400 examples with different r values). In Figure 7.11 the C4.5's competence map is reported.

From Figure 7.11 it appears that the error is, in most regions, very low, confirming the known robustness of C4.5. However, a failure region (error equal or greater than 20%) is observed as the term length ℓ takes on medium values ($\ell \in [5, 10]$), whenever the number n of variables is non-negligible ($n > 15$). It is no surprise that the learning difficulty increases with the total number n of variables, since the representativity of the training set (fixed at 200 positive and 200 negative examples, in the case of Figure 7.11) decreases. The relationship between the error and the term length ℓ is less obvious: for fixed m, r, and k the error first increases, and then decreases, as ℓ increases. However, it is almost insensitive to the imbalance ratio r.

The initial increase in the error with ℓ can be attributed to the myopic search of C4.5, greedily optimizing the information gain ratio criterion. Indeed, as the term length increases, each literal becomes less discriminant; furthermore, it is often the case that both a variable and its negation appear in some terms of the target concept, causing the information gain ratio criterion to miss the variables that contribute to the target concept. Therefore, a significant amount of look-ahead would be necessary to prevent the greedy search from becoming trapped

in local optima owing to erroneous early choices. In other words, the (univariate) gain ratio becomes a noisy selection criterion as the target concept involves more specific terms; hence the probability of making no errors along ℓ selections, based on this criterion, decreases with ℓ.

When the term length ℓ increases again ($\ell > 10$), the error decreases. This empirical finding was unexpected since the learning difficulty is usually seen as proportional to the target concept complexity, and ℓ is considered a factor in the complexity. The fact that the failure region does not depend much on the imbalance ratio r is unexpected too, since imbalanced example distributions are widely acknowledged as a factor in complexity.

The tentative interpretation offered by the authors for this finding is based on the observation that rules produced by C4.5 are not arbitrarily long, because they must cover a significant number of positive training examples; on average, their size is limited by a maximum value ℓ_c. Therefore, the probability $\mathrm{p}(n, \ell)$ for a leaf in a C4.5 tree to be irrelevant (to differ by at least one irrelevant literal from a generalization of a true conjunct) when learning a (k, ℓ)-term DNF concept is bounded by the probability that at least one irrelevant literal will be selected out of ℓ_c choices. However, the probability that an irrelevant feature will be selected decreases as ℓ increases.

The rise in the error as ℓ increases up to ℓ_c is thus explained as due to the increasing number of choices (hence the probability of error); the error falls for $\ell > \ell_c$ because it is the product of ℓ_c factors that all decrease as ℓ increases.

7.2.4 k-term DNF learning

The learning of k-term DNF concepts is an important task in machine learning. We have already seen, in the previous section, that such learning is mediated by the learning of decision trees. A direct approach was taken by Rückert *et al.* (2002), who showed that phase transitions exist even in this propositional setting. Moreover, they were able to locate and characterize these phase transitions using as parameters the number of terms k, the numbers $m_p = |\mathcal{P}|$ and $m_n = |\mathcal{N}|$ of positive and negative training examples, and the number n of Boolean variables. In the experiments, positive and negative examples of the target concept were generated by choosing either $x_i = 1$ or $x_i = 0$ with the same probability for each variable x_i $(1 \leqslant i \leqslant n)$. The search costs were measured by counting the number of partitionings generated by a complete algorithm.

The probability P_{sol} that a problem instance is soluble and the search costs were computed for a broad range of problem settings. Some results are given in Figure 7.12. In this figure the number of positive examples and k were kept constant, whereas the number of negative examples and of variables were varied.

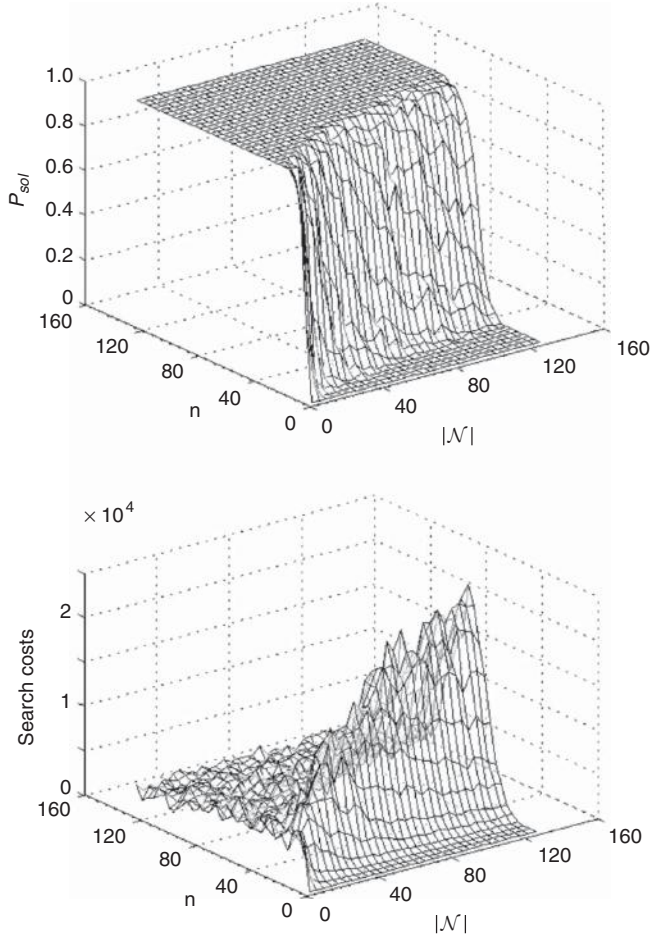


Figure 7.12 The probability that a problem instance is soluble, P_{sol} (upper panel), and the search costs (lower panel), plotted respectively as a three-dimensional graph and a contour plot for problem settings with $k = 3, m_p = 15, 1 \leqslant m_n \leqslant 128$, and $1 \leqslant n \leqslant 128$. Each data point represents the average over 100 problem instances. Reprinted from Rückert *et al*. (2002) with permission.

As expected, the search costs are especially high in the region of $P_{sol} \simeq 0.5$. In order to locate the phase transition, the authors took n as the control parameter, whose critical value n_{cr} is defined from the following condition:

$$P_{sol}(n_{cr}) = 0.5.$$

The authors also investigated from an algorithmic point of view the use of stochastic local search (SLS) for k-term DNF learning. They compared several variants that first reduce k-term DNF to SAT and then apply well-known stochastic local search algorithms such as GSAT and WalkSAT. The experimental results indicate that WalkSAT is able to solve the largest fraction of hard problem instances (Rückert *et al.*, 2002; Rückert and Kramer, 2003).

7.2.5 Vector quantization

Vector quantization is an effective and frequently applied method for unsupervised learning. It allows a large amount of data to be compressed into just a few prototypes, thus uncovering the structure underlying the data. Witoelar *et al*. (2008) applied methods from statistical mechanics, similar to those used in the analysis of neural networks, to off-line vector quantization with a rank-based cost function. The main finding was that phase transitions emerge in the training process; in fact, a critical number of examples is required to uncover the underlying structure.

More precisely, the authors considered a set of m vectors $\mathcal{S} = \{\vec{\mathbf{x}}^{(i)} \in \mathbb{R}^n | \ 1 \leqslant i \leqslant m\}$, drawn independently from a given probability distribution, and a set of k prototypes $\mathcal{Z} = \{\vec{\mathbf{z}}^{(j)} \in \mathbb{R}^n | \ 1 \leqslant j \leqslant k\}$. Using a squared Euclidean distance $d(\vec{\mathbf{x}}, \vec{\mathbf{y}}) = (\vec{\mathbf{x}} - \vec{\mathbf{y}})^2$, a *rank function* $g(r_j)$ can be defined such that, for every prototype $\vec{\mathbf{z}}^{(j)}$ and input vector $\vec{\mathbf{x}}$, r_j is the rank of $\vec{\mathbf{z}}^{(j)}$ with respect to its distance from $\vec{\mathbf{x}}$:

$$r_j = k - \sum_{h=1, h \neq j}^{k} \Theta_{h,j}, \tag{7.25}$$

$$\Theta_{h,j} = \mathbf{1}[d(\vec{\mathbf{x}}, \vec{\mathbf{z}}^{(h)}) - d(\vec{\mathbf{x}}, \vec{\mathbf{z}}^{(j)})]; \tag{7.26}$$

here $\mathbf{1}(u)$ is the step function, which is equal to 1 if $u \geqslant 0$ and to 0 otherwise. The rank function is defined as follows:

$$g(r_j) = \frac{e^{-r_j/\lambda}}{\sum_{j=1}^{k} e^{-r_j/\lambda}}, \tag{7.27}$$

where λ controls a "soft" assignment of the vectors to the prototypes (soft in that each vector may belong to more than one cluster). If $\lambda \to 0$, only the closest prototype is taken into account and so each vector is assigned to a unique prototype.

In the thermodynamic limit ($n \to \infty$) the number of examples is also assumed to increase linearly, namely, $m \sim n$. In the statistical physics approach, training is considered as a minimization process on the data $\mathcal{S}$ of the functional

$H(\vec{\mathbf{z}}^{(j)}, \ldots, \vec{\mathbf{z}}^{(k)})$, where:

$$H(\vec{\mathbf{z}}^{(1)}, \ldots, \vec{\mathbf{z}}^{(k)}) = \sum_{i=1}^{m} \varepsilon(\vec{\mathbf{z}}^{(1)}, \ldots, \vec{\mathbf{z}}^{(k)}, \vec{\mathbf{x}}^{(i)}), \tag{7.28}$$

$$\varepsilon(\vec{\mathbf{z}}^{(1)}, \ldots, \vec{\mathbf{z}}^{(k)}, \vec{\mathbf{x}}) = \frac{1}{2} \sum_{j=1}^{k} d(\vec{\mathbf{z}}^{(j)}, \vec{\mathbf{x}}) g(r_j) - \frac{1}{2} \vec{\mathbf{x}}^2. \tag{7.29}$$

As thermodynamic equilibrium is reached, each configuration $(\vec{\mathbf{z}}^{(1)}, \ldots, \vec{\mathbf{z}}^{(k)})$ is observed to have a probability given by the Gibbs distribution (see Chapter 2):

$$\mathrm{P}(\vec{\mathbf{z}}^{(1)}, \ldots, \vec{\mathbf{z}}^{(k)}) = \frac{1}{Z} e^{-\beta H(\vec{\mathbf{z}}^{(1)}, \ldots, \vec{\mathbf{z}}^{(k)})}; \tag{7.30}$$

Z is the partition function and $\beta = k_B T$. As described in Chapter 2, thermal averages $\langle \cdot \rangle$ are evaluated with respect to $\mathrm{P}(\vec{\mathbf{z}}^{(1)}, \ldots, \vec{\mathbf{z}}^{(k)})$ and can be obtained from derivatives of the free energy. The quantities introduced so far have been computed on a specific data set $\mathcal{S}$. In order to obtain generic properties, a new average, with respect to the data distribution (the quenched disorder), has to be performed; thus the *quenched* free energy may be obtained. The complete calculation requires the use of the replica trick. The authors resort to a simplified situation, one in which $\beta \to 0$ (the high-temperature limit). In this case, the noise introduced by the high temperature has to be compensated by the use of a large number of training examples, namely $\tilde{\alpha} = \beta m / n$.

It may be shown (Witoelar *et al.*, 2008) that the generalization error $\langle \varepsilon \rangle$ can be expressed as a function of two order parameters,

$$R_{ji} = \vec{\mathbf{z}}^{(j)} \cdot \vec{\mathbf{b}}^{(i)} \qquad \text{and} \qquad Q_{ji} = \vec{\mathbf{z}}^{(j)} \cdot \vec{\mathbf{z}}^{(i)},$$

where the $\vec{\mathbf{b}}^{(i)}$ $(1 \leqslant i \leqslant k)$ are the centers of the "true" clusters.

The behavior of the order parameter R_{j1} is shown in Figure 7.13. If $\lambda = 0$ (partitioning clustering) and $k = 2$, the structure in the data cannot be discovered if $\tilde{\alpha}$ is smaller than a critical value $\tilde{\alpha}_{cr}$. Above $\tilde{\alpha}_{cr}$ the structure emerges, and each prototype overlaps with exactly one cluster (see Figure 7.13(a)). The corresponding graph for $\langle \varepsilon \rangle$ is given in Figure 7.13(b). At the critical point, the graph is not differentiable but it is continuous in value. The finding for $k \geqslant 3$ are different; then, both R_{j1} and the $\langle \varepsilon \rangle$ graphs show a discontinuity in correspondence with the critical value α_{cr} (see Figures 7.14(a), (b)). The main conclusion of the work is that, for any k, no optimization strategy, however good, can uncover the structure in the data unless a sufficiently large number of examples is supplied.

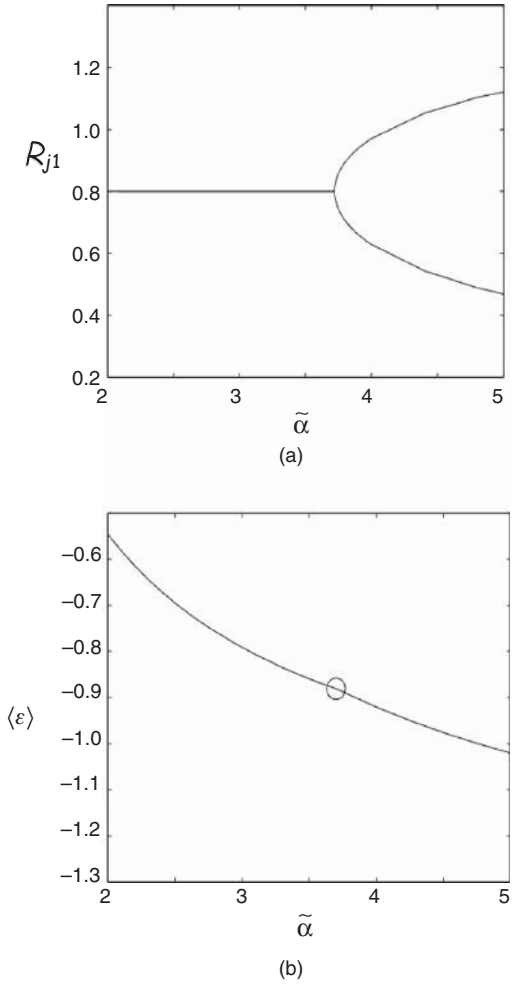


Figure 7.13 (a) Order parameter R_{j1} $(1 \leqslant j \leqslant 2)$ of the stable configuration for $k = 2$ vs. $\tilde{\alpha}$. There is a continuous phase transition at the critical value $\tilde{\alpha}_{cr}(2) \simeq 3.70$. (b) Graph of the mean error $\langle\varepsilon\rangle$ vs. $\tilde{\alpha}$ for $k = 2$. There is a discontinuity in the derivative at $\tilde{\alpha}_{cr}$. Reprinted from Witoelar *et al.* (2008) with permission.

7.3 Relational learning

The first results on the emergence of a phase transition in symbolic learning were reported by Giordana and Saitta and co-workers (Botta *et al.*, 1999; Giordana and Saitta, 2000; Giordana *et al.*, 2000a, b; Botta *et al.*, 2000, 2003; Serra *et al.*, 2001). They found that, in relational learning, the probability that an example is

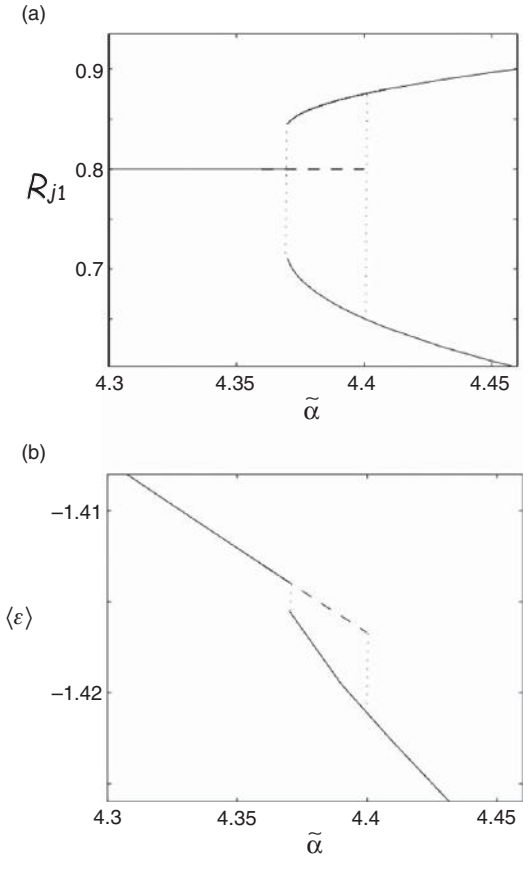


Figure 7.14 (a) Order parameter R_{j1} $(1 \leqslant j \leqslant 3)$ vs. $\tilde{\alpha}$. Two of the three R_{j1} coincide on the upper curve. The transition is first order and $\tilde{\alpha}_{cr}(3) \simeq 4.37$. (b) Graph of the mean error $\langle\varepsilon\rangle$, vs. $\tilde{\alpha}$ for $k = 3$. There is a discontinuity at $\tilde{\alpha}_{cr}$. Reprinted from Witoelar *et al*. (2008) with permission.

covered by a hypothesis (the order parameter), expressed in a DATALOG language, shows a steep transition from almost 1 (in the YES region) to almost 0 (in the NO region) with respect to certain control parameters that characterize both the hypothesis and the example. The analysis performed and the results obtained will be described in detail in Chapter 9.

Later on, the same authors investigated the impact of the presence of a phase transition in matching on the very feasibility of relational learning. They found that, around the phase transition, a wide region exists where learning proves to be impossible. A detailed description of this work will be reported in Chapter 10.

Following the first results, several other research groups have pursued the issue, extending and/or refining the original results of Giordana and Saitta. For instance, Wieczorek *et al.* (2006) replicated the experiments described by Giordana and Saitta (2000), introducing a *partial subsumption test* π instead of θ-subsumption and a *subsumption index* I_π.

More precisely, given a hypothesis h and an example e containing distinct variables, one says that h "π-*subsumes*" e iff there exists a subexpression h' in h ($h' \subseteq h$) and a substitution θ of the variables in h such that $h'\theta \subseteq e$. Moreover, given a hypothesis h and an example e such that h θ-subsumes e, the *subsumption index* I_π is a function from (h, e) to the interval [0, 1] that quantifies the "degree of subsumption" from h to e, i.e., the covering degree of h relative to h, with $h' \subseteq h$. If $h' = h$ then h θ-subsumes e and $I_\pi = 1$. When no subexpression h' θ-subsumes e, we have $I_\pi = 0$.

By using the above notion of partial subsumption, Wieczorek *et al.* (2006) found that the probability that a match exists (in a random set of pairs (h, e)) is still almost 1 on one side of the phase transition and smoothly descends towards 0 on the other side instead of jumping down to 0 abruptly. This finding may be relevant in designing more effective heuristics to learn long concepts.

An extensive study of phase transition emergence in relational learning has been made by Alphonse and Osmani (2007, 2008a, b, 2009). These authors started from Giordana and Saitta (2000) and Botta *et al.* (2003) and noticed that these results were obtained mostly by FOIL, which uses a top-down (TD) generate-and-test (GT) search strategy where refinements are based on the structure of the hypothesis space and the learning data is ignored. Therefore, Alphonse and Osmani (2007) suggested that data-driven (DD) strategies based on a generalization of a *seed* example may allow the pruning of irrelevant branches by the use of training data without relying on the evaluation function, thus possibly overcoming the problem of plateaus. Notably, *near-miss* examples are particularly effective: a top-down data-driven learner (TD–DD) can cross plateaus and reach the target concept whenever near misses are supplied in the training set, whereas these same near misses do not change the plateau profile and hence do not guide a TD–GT learner (Alphonse and Osmani, 2008b). Actually, substantially similar results to those with FOIL were obtained by Giordana and Saitta (2000) via G-Net, which is a data driven bidirectional searcher. Clearly, near misses would greatly ease learning but their availability may be problematic.

Near misses are defined, in this context, as negative examples that differ by only one literal from the seed example.

In order to perform a systematic analysis of the impact of plateaus on heuristic search in relational learning, Alphonse and Osmani designed a *consistency problem generator*, RLPG (relational learning problem generator) based on the model RB that was proposed for CSPs (see Chapter 4). In the model RLPG$(k, n, \alpha, N, |\mathcal{P}|, |\mathcal{N}|)$, the parameters k, n, α are the same as in model RB

(Xu and Li, 2000), whereas the definition of N is that of Giordana and Saitta (2000) (see Chapter 9); $|\mathcal{P}|$ and $|\mathcal{N}|$ are the numbers of positive and negative examples in the learning set, respectively. Using RLPG's properties, Alphonse and Osmani proved that the current hypothesis size, evaluated during learning, is a control parameter of the phase transition of the subsumption test. This result asymptotically guaranties the existence of a plateau for the heuristic search. Moreover, the size of the plateau is proven to grow sub-quadratically with the problem size. It was shown that problems of very small size can be generated in which the existence of plateaus is still guaranteed thus providing a well-suited benchmark for relational learning. Empirical confirmation was obtained by running several complete-search learners on problems generated by RLPG that exhibit the pathological case where informed-search learners degenerate into non-informed-search learners.

Alphonse and Osmani also performed an extensive study of the learning cost of several learners on inherently easy and hard instances of the corresponding consistency problem, looking for the typical easy–hard–easy pattern across the phase transition line. According to Gottlob *et al.* (1997), the simple bounded inductive logic programming (ILP) consistency problem is Σ_2-complete, i.e., it belongs to a class higher in the polynomial hierarchy than **NP**-complete problems. Since a conjecture has been put forward that a phase transition could be exhibited further up the polynomial hierarchy, the phase transition framework could be useful for investigating other PSPACE (see Garey and Johnson, 1979) problems as well. Actually, Alphonse and Osmani proved that this conjecture is true for the bounded ILP consistency problem, which is representative of relational learning.

Using the generator previously mentioned, Alphonse and Osmani (2009) generated and handled a set of benchmark datasets with controlled complexity, with the number of examples as control parameter. A first outcome of the work is that all well-known top-down relational algorithms, rooted in either the generate-and-test or the data-driven paradigm, fail to exhibit the standard "easy–hard–easy" pattern. Their complexity tends to increase with the number of examples, and therefore they exhibit an "easy–hard–hard" pattern. An exception is the depth-first bottom-up data driven (DF-BDD) algorithm, an lgg-based learner, which does not perform as well as the others on the "solvable" side of the phase transition but does perform well elsewhere.

7.3.1 Sequence learning

Sequences constitute a type of data that is very relevant in practical applications; they can be considered as an intermediate case between propositional and relational representation. It is then interesting to ascertain whether discontinuous

phenomena such as phase transitions also occur when one is inferring sequence models from data. Cornuéjols and co-workers (Pernot *et al.*, 2005) investigated learning of a specific type of model, namely *grammars*, inferred from strings of symbols. The approach followed by these researchers and the results obtained will be discussed in detail in Chapter 11.

7.4 Comments

The emergence of a phase transition, a typical phenomenon in many-body systems, appears in a large variety of learning approaches. Among these, neural networks are primary candidates for being handled as complex physical systems. In fact, it is quite natural to let neurons correspond to the elementary objects in a complex network and synaptic connections to interactions. By exploiting the notion of distance in a graph we can say that both in the single-layer perceptron and in the multi-layer perceptron, interactions are short-range, as they connect only neurons in adjacent layers.

A fundamental control parameter in different learning approaches appears to be the number of training examples. If the learner sees a critical number of them, it is as if it has seen them all. The training examples are drawn from a fixed (but usually unknown) distribution, and they act as a quenched disorder.

Surprising as it might be, analogous phenomena occur in learners that appear to be quite far from physical systems, such as symbolic learners. In Chapters 9–11 tentative explanations of why this happens will be suggested.

8

Learning, SAT, and CSP

Contents

An interesting issue, relevant to the main theme of this book, is whether, and if so how, learning, either propositional or relational, can be linked to the satisfiability (SAT) problem or to the constraint satisfaction problem (CSP). Establishing such a link can have important consequences in practice, because the theoretical aspects of SAT and CSP could be transferred to learning problems, and so the very effective algorithms developed for those combinatorial problems could be applied to learning tasks.

Intuitively, one may expect that propositional learning should be related to SAT and relational learning to CSPs. This intuition will be substantiated in the following sections. Moreover, finite CSPs must be reducible to SAT problems.

8.1 Reducing propositional learning to SAT

The formal transformation of a propositional learning problem into a SAT problem was investigated, for the first time, by Kamath *et al.* (1992) in relation to the induction of k-DNF (disjunctive normal form) rules. The same transformation was then applied by Rückert *et al.* (2002) to analyze the emergence of a phase transition in the corresponding SAT problem. Here, we will briefly review the

transformation proposed by Kamath, then we will discuss the implications and the extension to propositional learning in general.

A k-DNF concept definition consists of a set of k propositional rules with the format

$$\gamma_1 \vee \gamma_2 \vee \cdots \vee \gamma_k \rightarrow h, \tag{8.1}$$

In an equivalent way, rule (8.1) can be written as a set of k Horn clauses: $\gamma_1 \rightarrow h, \ldots, \gamma_k \rightarrow h$.

$\gamma_1, \gamma_2, \ldots, \gamma_k$ being conjunctions of conditions on the attribute values describing the instance to be classified.

The problem of learning a k-DNF concept definition from a learning set of concept instances, classified by a teacher, was formally introduced by Valiant (1984) as a fundamental problem in computational learning theory. Since then it has been investigated by many authors. In fact, this form of concept definition is relevant to many applications but involves a learning problem that is computationally hard (Kearns and Vazirani, 1994).

Kamath *et al.* (1992) approached the problem of deriving a k-DNF rule, complete and consistent with respect to a learning set, by starting from a *rule template* defined *a priori*. The assumption was that the learning instances are described by Boolean attributes only (i.e., attributes may only have the values 1 or 0). However, this assumption is not limitative because any multiple-valued attribute can always be transformed into an equivalent set of Boolean attributes.

Let $\vec{\mathbf{x}}$ denote the n-dimensional vector of Boolean attributes describing a learning instance. Moreover, let x_j $(1 \leqslant j \leqslant n)$ denote the ith component of $\vec{\mathbf{x}}$. A k-DNF rule can be represented as an expression in Boolean algebra, where the n binary attributes are the input variables. Rules of this form have an immediate representation in terms of logical circuits, which provide an intuitive graphical representation (see Figure 8.1).

The rule template has the structure of a k-DNF rule over the whole set of variables extended with a set of control variables, which are used to make the template consistent with the positive and negative learning instances by the selection or exclusion of single components of the input variables in $\vec{\mathbf{x}}$. Then, the problem of learning a k-DNF concept definition is reduced to the problem of finding a consistent assignment to the control variables, and this, in turn, can be reduced to a SAT problem.

The rule template (see Figure 8.1) is the OR of k conjunctions, where each conjunction γ_i $(1 \leqslant i \leqslant k)$ is the AND of n variables y_{ij} $(1 \leqslant i \leqslant k, 1 \leqslant j \leqslant n)$, each obtained by combining a Boolean attribute x_j with a pair of two control variables $\langle s_{ij}, s'_{ij} \rangle$. Depending on the values of s_{ij} and s'_{ij}, either the variable y_{ij} reproduces the value of x_j or of $\overline{x}_j$ or it becomes independent of x_j. More specifically, the logical expression defining y_{ij} is the following:

$$y_{ij} = (x_j \vee s_{ij}) \wedge (\overline{x}_j \vee s'_{ij}). \tag{8.2}$$

The truth values of y_{ij} are given in Table 8.1.

Table 8.1 Truth table corresponding to the variable y_{ij} defined by (8.2)

s_{ij}	s'_{ij}	y	Comment
0	1	$y_{ij} = x_j$	$y_{ij} \equiv x_j$
1	0	$y_{ij} = \overline{x}_j$	$y_{ij} \equiv \overline{x}_j$
1	1	1	$y_{ij} \equiv$ *true*, i.e., x_j is irrelevant
0	0	0	$y_{ij} \equiv$ *false*, i.e., a contradiction occurs

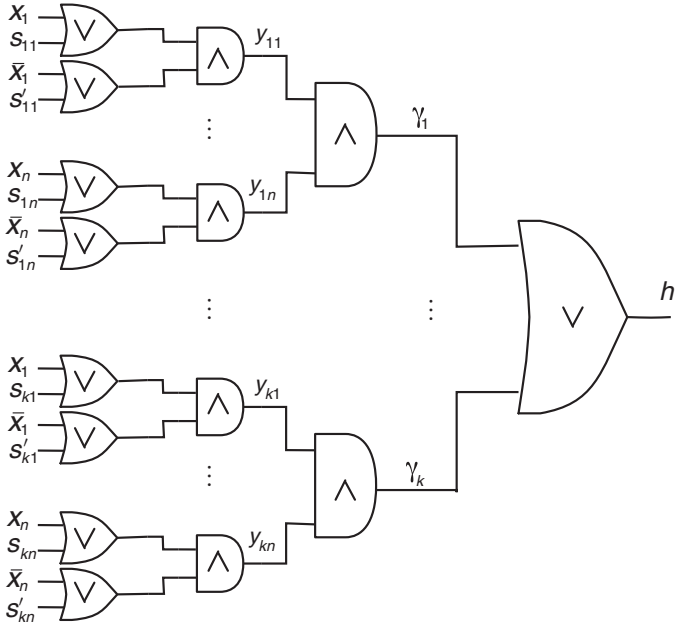


Figure 8.1 Logical circuit corresponding to the rule template. The circuit is then generalized in order that it can learn a k-DNF classification rule.

Setting $s_{ij} = 1$ and $s'_{ij} = 1$ is equivalent to applying the "dropping condition" rule widely used in machine learning algorithms.

From this table we note that when $s_{ij} = 0$ and $s'_{ij} = 1$ the value of y_{ij} corresponds to the value of x_j, while it corresponds to $\overline{x}_j$ when $s_{ij} = 1$ and $s'_{ij} = 0$. Moreover, if $s_{ij} = 1$ and $s'_{ij} = 1$ then y_{ij} does not depend on x_j as it is always *true* (i.e., $y_{ij} = 1$).

Therefore the control variables s_{ij}, s'_{ij} provide a tool for selecting a condition on the corresponding attribute x_j or dropping the dependency upon x_j. Finally we notice that by setting $s_{ij} = 0$ and $s'_{ij} = 0$ we obtain $y_{ij} = x_j \wedge \overline{x}_j$, which is always *false*, causing γ_i to be *false* as well. Thus, this setting must be forbidden for any pair $\langle s_{ij}, s'_{ij} \rangle$. This can be stated by requiring that the clauses in the

following set, Γ_1, must all be satisfied:

$$\Gamma_1 = \{(s_{i,j} \vee s'_{ij}) \mid 1 \leqslant i \leqslant k, 1 \leqslant j \leqslant n\}. \tag{8.3}$$

Expression (8.3) defines the initial set of nk clauses of the corresponding SAT problem.

We will now consider the constraints due to the learning set $\mathcal{S}_L$. Every positive or negative learning instance $e \in \mathcal{S}_L$ will impose some constraints on the output value h of the k-DNF rule to be learned. By propagating these constraints toward the inputs, new clauses on the control variables are obtained. First let us consider the negative examples. For any example $e_r^- \in \mathcal{N}$ the value of h must be *false* (i.e., $h = 0$). This means that all the conjunctions $\gamma_i^{(r)}$ must have the value 0 for that example. This condition can be satisfied if, for every $\gamma_i^{(r)}$, at least one variable $y_{ij}^{(r)}$ has the value 0, i.e., $\exists t | \overline{y}_{it}^{(r)} = 1$. All these conditions can be grouped together in the following set Γ'_2 of clauses:

$$\Gamma'_2 = \left\{ \bigvee_{j=1}^{n} \overline{y}_{ij}^{(r)} \;\middle|\; 1 \leqslant r \leqslant |\mathcal{N}|, 1 \leqslant \mathrm{i} \leqslant k \right\}, \tag{8.4}$$

which must be verified on each example $e_r^- \in \mathcal{N}$. According to definition (8.2) of y_{ij}, and assuming a specific value of x_j for a learning instance $e_r^- \in \mathcal{N}$, the clauses in (8.4) can be modified in order to obtain constraints on the control variables s_{ij} and s'_{ij}. More specifically, if attribute $x_j^{(r)} = 0$ in e_r^- then we obtain from (8.2) that $y_{ij}^{(r)} = s_{ij}$. Thus, in order to have $\overline{y}_{ij}^{(r)} = 1$ we must have $s_{ij} = 0$. As a consequence we may substitute the literal $\overline{y}_{ij}^{(r)}$ by the literal $\overline{s}_{ij}$ in (8.4). Applying analogous reasoning to the case $x_j^{(r)} = 1$ in e_r^-, we can conclude that in this case the literal $\overline{y}_{ij}^{(r)}$ in (8.4) can be replaced by the literal $\overline{s}'_{ij}$. Thus we obtain the following set of clauses:

$$\Gamma_2 = \left\{ \bigvee_{j=1}^{n} \overline{\sigma}_{ij}(x_j^{(r)}) \;\middle|\; 1 \leqslant r \leqslant |\mathcal{N}|, 1 \leqslant \mathrm{i} \leqslant k \right\}, \tag{8.5}$$

where

$$\sigma_{ij}(x_j^{(r)}) = \begin{cases} s_{ij} & \text{if } x_j^{(r)} = 0, \\ s'_{ij} & \text{if } x_j^{(r)} = 1. \end{cases} \tag{8.6}$$

We may notice that in (8.4) the $y_{ij}^{(r)}$ depend on the example e_r^-, whereas s_{ij} and s_{ij} do not. However, the dependence of the clauses in Γ_2 on a specific example is reflected in the different number of s_{ij} and s_{ij} occurring in each clause and in their position. In conclusion, for every $e_r^- \in \mathcal{N}$ a set of k clauses can be derived

from expression (8.4) by applying the above reasoning. Then $|\Gamma_2| = k|\mathcal{N}|$, and the set Γ_2 is added to the SAT problem.

Let us now consider the constraints due to the positive examples $e^+ \in \mathcal{P}$. Every $e_r^+ \in \mathcal{P}$ must satisfy the k-DNF rule, requiring that $h = 1$. This condition is achieved if, for any positive example, at least one conjunction γ_i ($1 \leqslant i \leqslant k$) is verified. This condition can be stated by defining, for every example $e_r^+ \in \mathcal{P}$, a set $Z_r = \{z_i^{(r)} \mid 1 \leqslant i \leqslant k\}$ of k variables. The variable $z_i^{(r)}$ assumes the value 1 iff the conjunction γ_i is true in e_r^+. Then the requirement that at least one conjunction is true in each e_r^+ can be translated into the following set Γ_3' of clauses:

$$\Gamma_3' = \left\{ \bigvee_{i=1}^{k} z_i^{(r)} \;\middle|\; 1 \leqslant r \leqslant |\mathcal{P}|, 1 \leqslant i \leqslant k \right\}. \tag{8.7}$$

Again, in order to be operational we must link the variables in Z_r to the control variables, as we did for the negative examples. To this end we observe that requiring $z_i^{(r)}$ to be 1 entails $y_{ij}^{(r)} = 1$ ($1 \leqslant j \leqslant n$). Formally this can be stated as follows:

$$z_i^{(r)} \rightarrow \left(y_{i1}^{(r)} \wedge y_{i2}^{(r)} \wedge \cdots \wedge y_{in}^{(r)} \right). \tag{8.8}$$

We remember that $z_i^{(r)} \rightarrow (y_{i1}^{(r)} \wedge y_{i2}^{(r)} \wedge \cdots \wedge y_{in}^{(r)})$ is equivalent to $\overline{z}_i^{(r)} \vee \left(y_{i1}^{(r)} \wedge y_{i2}^{(r)} \wedge \cdots \wedge y_{in}^{(r)} \right)$. Then, by applying the distributive property of the AND operator, expression (8.8) can be rewritten in the equivalent form

$$\bigwedge_{j=1}^{n} (\overline{z}_i^{(r)} \vee y_{ij}^{(r)}). \tag{8.9}$$

Applying reasoning analogous to that for the negative learning instances, the variables $y_{ij}^{(r)}$ can be tied to the control variables s_{ij} and s'_{ij} according to the value of $x_i^{(r)}$ in any single positive example. In this case, we want $y_{ij}^{(r)}$ to be *true*; then the variable $y_{ij}^{(r)}$ in (8.9) is replaced by s_{ij} or by s'_{ij} according to whether the value of $x_j^{(r)}$ is 0 or 1. Therefore, the set Γ_3' becomes the following:

$$\Gamma_3 = \{(\overline{z}_i^{(r)} \vee \sigma_{ij}(x_j^{(r)}) | 1 \leqslant r \leqslant |\mathcal{P}|, 1 \leqslant i \leqslant k, 1 \leqslant j \leqslant n\}, \tag{8.10}$$

where $\sigma_{ij}(x_j^{(r)})$ is the function defined by (8.6). The cardinality of Γ_3 is $kn|\mathcal{P}|$. Then, by collecting the clauses in the three sets Γ_1, Γ_2, and Γ_3, we obtain c clauses, where

$$c = nk + k|\mathcal{N}| + kn|\mathcal{P}|. \tag{8.11}$$

The number v of variables is

$$v = 2nk + k|\mathcal{P}| = k(2n + |\mathcal{P}|). \tag{8.12}$$

Then the parameter $\alpha = c/v$ assumes the value

$$\alpha(|\mathcal{N}|, |\mathcal{P}|, n) = \frac{c}{v} = \frac{nk + k|\mathcal{N}| + kn|\mathcal{P}|}{k(2n + |\mathcal{P}|)} = \frac{|\mathcal{N}| + n(|\mathcal{P}| + 1)}{2n + |\mathcal{P}|}. \tag{8.13}$$

Notice that the resulting SAT has clauses of varying length: Γ_1 and Γ_3 contain clauses of length 2 whereas Γ_2 contains clauses of length n. Thus it is not possible directly to transfer to k-DNF learning the results about the location of the phase transition. Nonetheless, from equation (8.13) it can be seen that

$$\lim_{|\mathcal{N}|\to\infty} \alpha(|\mathcal{N}|, |\mathcal{P}|, n) = \infty,$$
$$\lim_{|\mathcal{P}|\to\infty} \alpha(|\mathcal{N}|, |\mathcal{P}|, n) = n.$$

From the above expressions one can see that increasing only the number of negative examples makes the SAT problem unsatisfiable. Increasing only the number of positive examples allows α to converge to the value n; in this case, α scales linearly with n.

—— EXAMPLE ——

Suppose that we want to learn a 2-DNF rule classifying objects described by three Boolean variables x_1, x_2, and x_3. The learning set $\mathcal{S}_L$ contains the six examples, three positive and three negative, shown below in Figure 8.2. There are eight examples in all, so generalization is still possible because there are two unclassified examples. In order to find the first set of clauses

x_1	x_2	x_3	h	
0	0	0	1	
0	1	0	1	Positive example set $\mathcal{P}$
0	0	1	1	
1	0	0	0	
1	1	0	0	Negative example set $\mathcal{N}$
0	1	1	0	

Figure 8.2 Learning set used to acquire a 2-DNF formula.

in the corresponding SAT problem, we must define the following set of 12 variables:

$$\{s_{11}, s'_{11}, s_{12}, s'_{12}, s_{13}, s'_{13}, s_{21}, s'_{21}, s_{22}, s'_{22}, s_{23}, s'_{23}\}.$$

The rule template for the 2-DNF rule is then $h = \gamma_1 \vee \gamma_2$, with

$$\gamma_1 = (x_1 \vee s_{11}) \wedge (\overline{x}_1 \vee s'_{11}) \wedge (x_2 \vee s_{12}) \wedge (\overline{x}_2 \vee s'_{12}) \wedge (x_3 \vee s_{13}) \wedge (\overline{x}_3 \vee s'_{13}),$$

$$\gamma_2 = (x_1 \vee s_{21}) \wedge (\overline{x}_1 \vee s'_{21}) \wedge (x_2 \vee s_{22}) \wedge (\overline{x}_2 \vee s'_{22}) \wedge (x_3 \vee s_{23}) \wedge (\overline{x}_3 \vee s'_{23}).$$

The first set of clauses, defined by (8.3), is the following:

$$\Gamma_1 = \{(s_{11} \vee s'_{11}), (s_{12} \vee s'_{12}), (s_{13} \vee s'_{13}), (s_{21} \vee s'_{21}), (s_{22} \vee s'_{22}), (s_{23} \vee s'_{23})\}.$$

According to (8.5) a second set of clauses is obtained by instantiating the function (8.6) on the negative examples:

$$\begin{aligned}\Gamma_2 &= \{(\overline{s}_{11} \vee \overline{s}'_{12} \vee \overline{s}'_{13}), (\overline{s}_{21} \vee \overline{s}'_{22} \vee \overline{s}'_{23}),\\ &= \quad (\overline{s}_{11} \vee \overline{s}_{12} \vee \overline{s}'_{13}), (\overline{s}_{21} \vee \overline{s}_{22} \vee \overline{s}'_{23}),\\ &= \quad (\overline{s}'_{11} \vee \overline{s}_{12} \vee \overline{s}_{13}), (\overline{s}'_{21} \vee \overline{s}_{22} \vee \overline{s}_{23})\}.\end{aligned}$$

The clauses in Γ_2 are to be added to the SAT problem.

Finally, considering the positive examples, we must introduce the new variables:

$$z_1^{(1)}, z_2^{(1)}, z_1^{(2)}, z_2^{(2)}, z_1^{(3)}, z_2^{(3)}.$$

The last set of clauses is thus

$$\Gamma'_3 = \left\{(z_1^{(1)} \vee z_2^{(1)}), (z_1^{(2)} \vee z_2^{(2)}), (z_1^{(3)} \vee z_2^{(3)})\right\},$$

which can be transformed into:

$$\begin{aligned}\Gamma'_3 &= \Big\{(\overline{z}_1^{(1)} \vee y_{11}^{(1)}), (\overline{z}_1^{(1)} \vee y_{12}^{(1)}), (\overline{z}_1^{(1)} \vee y_{13}^{(1)}),\\ &= \quad (\overline{z}_2^{(1)} \vee y_{21}^{(1)}), (\overline{z}_2^{(1)} \vee y_{22}^{(1)}), (\overline{z}_2^{(1)} \vee y_{23}^{(1)}),\\ &= \quad (\overline{z}_1^{(2)} \vee y_{11}^{(2)}), (\overline{z}_1^{(2)} \vee y_{12}^{(2)}), (\overline{z}_1^{(2)} \vee y_{13}^{(2)}),\\ &= \quad (\overline{z}_2^{(2)} \vee y_{21}^{(2)}), (\overline{z}_2^{(2)} \vee y_{22}^{(2)}), (\overline{z}_2^{(2)} \vee y_{23}^{(2)},\\ &= \quad (\overline{z}_1^{(3)} \vee y_{11}^{(3)}), (\overline{z}_1^{(3)} \vee y_{12}^{(3)}), (\overline{z}_1^{(3)} \vee y_{13}^{(3)}),\\ &= \quad (\overline{z}_2^{(3)} \vee y_{21}^{(3)}), (\overline{z}_2^{(3)} \vee y_{22}^{(3)}), (\overline{z}_2^{(3)} \vee y_{23}^{(3)})\Big\}.\end{aligned}$$

Substituting the $y_{ij}^{(r)}$ by the s_{ij} and s'_{ij}, we finally obtain

$$\begin{aligned}\Gamma_3 &= \Big\{(\overline{z}_1^{(1)} \vee s'_{11}), (\overline{z}_1^{(1)} \vee s'_{12}), (\overline{z}_1^{(1)} \vee s'_{13}),\\ &= \quad (\overline{z}_2^{(1)} \vee s'_{21}), (\overline{z}_2^{(1)} \vee s'_{22}), (\overline{z}_2^{(1)} \vee s'_{23}),\\ &= \quad (\overline{z}_1^{(2)} \vee s'_{11}), (\overline{z}_1^{(2)} \vee s_{12}), (\overline{z}_1^{(2)} \vee s'_{13}),\\ &= \quad (\overline{z}_2^{(2)} \vee s'_{21}), (\overline{z}_2^{(2)} \vee s_{22}), (\overline{z}_2^{(2)} \vee s'_{23}),\\ &= \quad (\overline{z}_1^{(3)} \vee s'_{11}), (\overline{z}_1^{(3)} \vee s'_{12}), (\overline{z}_1^{(3)} \vee s_{13}),\\ &= \quad (\overline{z}_2^{(3)} \vee s'_{21}), (\overline{z}_2^{(3)} \vee s'_{22}), (\overline{z}_2^{(3)} \vee s_{23})\Big\}.\end{aligned}$$

The conjunction of all the clauses in $\Gamma_1 \cup \Gamma_2 \cup \Gamma_3$ defines the CNF formula characterizing the SAT problem. A possible solution to the problem is illustrated in Figure 8.3, which shows the template, the value for the

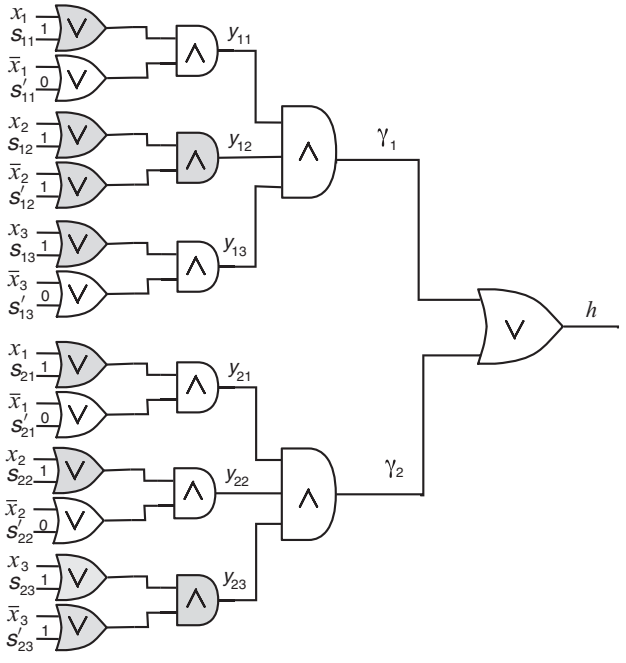


Figure 8.3 A possible solution of the learning problem described in the Example. In the template corresponding to the problem the final values of the control variables s_{ij}, s'_{ij} are stated on the corresponding input; the shaded components indicate attributes that have been eliminated from the final rule. The learned 2-DNF formula is $h = (\overline{x}_1 \wedge \overline{x}_3) \vee (\overline{x}_1 \wedge \overline{x}_2)$.

control variable s_{ij}, s'_{ij}, and the rule obtained after the template has been simplified by the removal of redundant literals. In conclusion, we have transformed the learning problem into a $(2+p)$-SAT with $c = 30$ clauses, $v = 18$ variables and ratio $\alpha = 1.667$. Of the clauses, 12 are of length 2 and 18 are of length 3; hence $p = 0.6$ (see Section 3.4).

8.2 Phase transitions and local search in propositional learning

As already mentioned, k-DNF concept descriptions are highly representative of propositional learning. In fact, they correspond to the standard output format of many learners that work in propositional logics, such as, for instance, RIPPER (Cohen, 1995), PFOIL (Mooney, 1995), and C4.5rules (Quinlan, 1993).

A commonly employed heuristic consists in trying to find the simplest possible solution, i.e., the solution with the smallest k. The fundamental difference between the approach used by classical learners and the problem reformulation described in the previous section is that in the former case k is obtained as an *a posteriori* result provided by the learner while in the latter case k is given *a priori*. However, from a practical point of view this is not restrictive, since many values of k can be tried as the learner searches for the smallest value for which a solution does exist. It is worth noticing (Rückert *et al.*, 2002; Rückert and Kramer, 2003) the following.

$k = |\mathcal{P}|$ means no generalization of the examples.

- If $k = |\mathcal{P}|$ then the problem always has a trivial solution, because every positive example can be encoded into a specific disjunct.
- If a solution exists for a given value k_s, a solution will exist for any $k > k_s$. Adding new disjuncts to an already true disjunctive formula does not affect its truth. Thus it is interesting to find the minimum value k_m for which a solution exists.
- Finding a solution for any $k > k_m$ will be easier than finding a solution for k_m, because the problem is less restrictive.

We will now consider two questions.

1. Given that a k-DNF learning problem can be mapped to a SAT problem, and that SAT problems typically exhibit a phase transition, does k-DNF learning exhibit a phase transition as well?
2. Apart from the theoretical interest, has the transformation from k-DNF learning to SAT any practical advantage? For instance, can the very effective search algorithms developed for SAT be successfully used for learning?

As mentioned in Chapter 7, the phase transition emergence in k-DNF learning problems has been throughly investigated by Rückert *et al.* (2002); they showed that a phase transition does exist for a critical value of the number n of variables (Boolean features). Moreover, the critical point n_{cr} of the phase transition was characterized as a function of k and of the numbers $|\mathcal{P}|$ and $|\mathcal{N}|$ of positive and negative training examples, respectively:

$$n_{cr} \approx [a(k) \log_2 |\mathcal{N}|]^{(b|\mathcal{P}|+c)}. \tag{8.14}$$

Formula (8.14) is a reasonable conjecture, which was experimentally validated by these authors. It is worth noticing that the phase transition analysis was done

using an ad hoc learning algorithm based on set covering, without exploiting the transformation into SAT.

In the same paper, and in a companion paper (Rückert and Kramer, 2003), an extensive comparison of different stochastic local search (SLS) algorithms was made. As none of these algorithms was designed to work directly on the k-DNF learning problem, this last was reduced to SAT (for which several such searchers exist) using the transformation of Kamath *et al.* (1992). The algorithms selected were GSAT (Selman *et al.*, 1992), GSAT + Random Walk, GSAT + Tabu, and WalkSAT (Kautz and Selman, 1996). Moreover, these authors designed an efficient native SLS algorithm (Rückert and Kramer, 2003), which is able to solve hard k-DNF learning problems without transforming them into SAT.

The experiments were performed on three sets each containing 100 hard solvable problems taken from the phase transition region. Each learning problem needed to be solved by finding a k-DNF formula. The reported results clearly show that the native SLS, explicitly designed to solve this specific task, dominates the algorithms designed for SAT; in fact, only WalkSAT reaches performances that, though inferior, are comparable with those of the native SLS.

From the machine learning side, three learners have been selected for comparison: C5.0 (an improved version of C4.5 (Quinlan, 1993)), RIPPER (Cohen, 1995), and PART (Witten and Frank, 2005). However, these learners are primarily aimed not to minimize the number of disjuncts but to reliably cover the class instances. Therefore, the solutions they provide typically contain more disjuncts than the minimum required and exhibit misclassification errors on the learning set itself because of the covering criterion used to guide the algorithms. Nevertheless they always do provide a solution, but one that has not been tested on an independent test set to evaluate its real accuracy. The computational complexity of the search algorithm is much lower for the machine learning algorithms than for the SAT algorithms.

Concept learning and constraint satisfaction have different goals.

The above results touch a crucial point. Searching for a concept description is not quite a problem of constraint satisfaction. In the former case, as discussed in Chapter 6, the primary goal is to find a formula that exhibits a low generalization error. Therefore, in general, completeness of the learning set is sacrificed in favor of other criteria related to predictiveness. On the contrary, SAT algorithms aim at finding, when possible, a solution that satisfies all constraints and is then complete and consistent with regard to the learning set. For this reason it is impossible to use pure SAT algorithms if there is noise (it is always present in real-world domains), because noise makes the learning set inconsistent; then, a pure SAT algorithm cannot find a solution. However, in practice algorithms for MaxSAT can be used for this task, as was done by Rückert *et al.* (2002).

Nevertheless, like the other algorithms, MaxSAT has the goal of maximizing the number of satisfied constraints, without regard for the effect this may have

on the predictiveness of the formula being constructed. This can lead to solutions that overfit the data and are poorly predictive when applied to data not belonging to the learning set. It is not obvious how to modify SAT or MaxSAT solvers' heuristics in order to fix this problem.

Finally, we note that the same problem remains even when the learning set is consistent: according to the learning heuristic it may be preferable to leave uncovered some examples in order to avoid data overfitting, whereas SAT or MaxSAT solvers would provide a complete and consistent solution.

From the above discussion it follows that, on the one hand, mapping a k-DNF learning problem to a SAT learning problem is not likely to systematically yield better practical results. On the other hand, the transformation has theoretical relevance as it helps one's understanding of the mechanisms underlying learning.

Induction of decision trees as a SAT problem

A more successful transformation from a decision-tree learning problem to SAT was described by Bessière *et al.* (2009), who formulated the problem as one of optimization, namely, to find the tree with the minimum number of nodes that correctly classifies all the examples in the learning set. They investigated a formulation based on satisfiability and constraint programming and compared their approach against standard induction algorithms. More precisely, let $\mathcal{T} = (X, U, r)$ be a binary tree, where X is the set of nodes, U is the set of edges, $r \in X$ is the root, and $L \subseteq X$ is the set of leaves. A *decision tree* based on $\mathcal{T}$ is a labeled tree in which each internal node x is labeled with a feature, taken from a set $\mathcal{F}$ and denoted $f(x)$. Each edge $(x, y) \in U$ is labeled with a Boolean function $g(x, y)$, where $g(x, y) = 0$ if y is the left child of x and $g(x, y) = 1$ if y is the right child of x. The size of the decision tree is the number of nodes of $\mathcal{T}$. Given a learning set $\mathcal{S}_L = \{e_1, \ldots, e_m\}$, the authors exhibited a SAT formula that is satisfiable iff there exists a decision tree, based on $\mathcal{T}$, that classifies $\mathcal{S}_L$ correctly.

The trees obtained through this reformulation were compared by Bessière *et al.* with those induced by standard state-of-the-art learners, both pruned and unpruned, on several publicly available datasets. The results clearly show that the constraint programming (CP) approach produces very accurate trees, which are smaller than those found using standard greedy unpruned methods. Even when compared with pruned trees, the accuracy of the CP approach is often competitive with, or exceeds, that of pruned trees built using standard methods.

8.3 The FOL covering test as a CSP

As introduced in Section 5.2.3, in most relational (ILP) learners the basic covering test consists in proving the satisfiability of existentially quantified

conjunctive formulas such as $\exists\vec{\mathbf{x}}\,[\varphi(\vec{\mathbf{x}})]$, with n variables (belonging to a given set $\mathbf{X} = \{x_1, \ldots, x_n\}$) and m literals (predicates or their negation from a set $\mathbf{P}$) over a universe U. The universe consists of the set of relations (tables) containing the extensions of the atomic predicates that are true for an example e; a tabular instance representation was given in Figure 5.9. The formula $\varphi(\vec{\mathbf{x}})$ is satisfiable if there exists at least one model (instance) of it in e. We will call the pair $(\varphi(\vec{\mathbf{x}}), e)$ a *matching problem*.

Recalling the definition of a CSP introduced in Chapter 4, it is clear that a matching problem can be mapped directly to a CSP: the n variables and their associated domains play the same role whereas the m relations expressed by the literals occurring in $\varphi(\vec{x})$ correspond to the set $\mathbf{R}$ of relations in a CSP. In learning relational concepts, each hypothesis (formula) $\varphi(\vec{x})$ generated by the learner must be matched to all the training examples, each corresponding to a different universe. Given that a learner may generate thousands of hypotheses during its work, the complexity of the matching problem may deeply affect the very feasibility of learning.

The equivalence between a matching problem and a CSP will be formalized and discussed in detail in Chapter 9.

8.4 Relation between CSP and SAT

In previous chapters we have seen that the classes of SAT problems and CSPs are characterized by the presence of a phase transition. In this section we will explore the relations between the two classes. Clearly such relations are guaranteed to exist, because both SAT and CSP are **NP**-complete and thus in principle can be translated one into the other in polynomial time. Before entering into details, we note that any finite DATALOG theory can always be translated into an equivalent set of sentences in propositional logic.

Translating CSP into SAT

Let us now introduce an algorithm for translating a finite CSP into an equivalent SAT problem. The fundamental difference between CSP and SAT is that variables in a CSP may have values in a large domain, whereas variables in SAT are Boolean, i.e., they may only assume the two mutually exclusive values $\{1, 0\}$. Thus we need a transformation for converting multi-valued variables in a CSP into a set of binary variables.

We will consider here only binary CSPs, as the extension to non-binary CSPs is immediate. Let $x_i \in \mathbf{X}$ be a variable in a CSP with domain D_i, with cardinality $|D_i| = d_i$. We define a set $\mathbf{B}_i$ of Boolean variables associated with the domain D_i. A variable $x_i{:}a_{ir}$, belonging to $\mathbf{B}_i$, assumes the value *true* when x_i is bound to the value $a_{ir} \in D_i$ and assumes the value *false* otherwise. Notice that one and only one variable in the set $\mathbf{B}_i$ must be *true*. These constraints can be encoded in

two sets of clauses, $\mathbf{S}_1$ and $\mathbf{S}_2$. The set $\mathbf{S}_1$ contains the clauses stating that each variable must assume at least one value in its domain:

$\mathbf{S}_1$ contains clauses stating that each variable must assume one of its values.

$$\mathbf{S}_1 = \{\alpha_i | 1 \leqslant i \leqslant n\}, \tag{8.15}$$

$$\alpha_i = \bigvee_{r=1}^{d_i} x_i{:}a_{ir}. \tag{8.16}$$

$\mathbf{S}_2$ contains clauses stating that each variable must assume only one of its values.

The set $\mathbf{S}_2$ contains clauses stating that each variable may assume only one value from its domain:

$$\mathbf{S}_2 = \{\beta_{rs}^{(i)} | 1 \leqslant i \leqslant n, 1 \leqslant r, s \leqslant d_i, r \neq s\}, \tag{8.17}$$

$$\beta_{rs}^{(i)} = \overline{x_i{:}a_{ir}} \vee \overline{x_i{:}a_{is}}. \tag{8.18}$$

We have now to translate the constraints in the set $\mathbf{R}$ (with cardinality m) of the CSP into clauses in SAT. With this aim, let us consider the matrices $M_{ij}^{(h)}$ introduced at the beginning of Chapter 4, which correspond to the constraints $R_h(x_i, x_j)$ $(1 \leqslant h \leqslant m)$. Each *false* entry in $M_{ij}^{(h)}$ corresponds to nogoods and must be excluded. In other words, if the entry $m_{rs}^{(h)}$ is *false* in $M_{ij}^{(h)}$ then the pair of assignments $(x_i = a_{ir})$ and $(x_j = a_{js})$ is forbidden. By translating this condition into the notation introduced before, we obtain a third set, $\mathbf{S}_3$, of clauses:

Clauses excluding the nogoods

$$\mathbf{S}_3 = \{\gamma_{rs}^{(h)} | 1 \leqslant h \leqslant m, m_{rs}^{(h)} = F\}, \tag{8.19}$$

$$\gamma_{rs}^{(h)} = \overline{x_i{:}a_{ir}} \vee \overline{x_j{:}a_{js}}. \tag{8.20}$$

Solving the CSP is then equivalent to solving the SAT problem that has the set of variables

$$\mathbf{B} = \bigvee_{i=1}^{n} \mathbf{B}_i$$

and the set of clauses

$$\mathbf{S} = \mathbf{S}_1 \cup \mathbf{S}_2 \cup \mathbf{S}_3.$$

Globally, the formula to be made true, corresponding to the obtained SAT, is the following:

$$\varphi = \left(\bigwedge_{i=1}^{n} \alpha_i \right) \wedge \left(\bigwedge_{i=1}^{n} \bigwedge_{r,s=1, r \neq s}^{d_i} \beta_{rs}^{(i)} \right) \wedge \left(\bigwedge_{h=1}^{m} \bigwedge_{r,s | m_{rs}^{(h)} = F} \gamma_{rs}^{(h)} \right). \tag{8.21}$$

In order to clarify the translation described above, we will use the small CSP shown in Figure 5.9.

EXAMPLE

Let $\mathbf{X} = \{x_1, x_2, x_3\}$ be a set of variables, all of which take values in the same domain $D = D_1 = D_2 = D_3 = \{a, b, c, d\}$, with cardinality $d = 4$. Moreover, let $\mathbf{R} = \{R_1, R_2, R_3\}$ be the set of three constraints (relations) represented in tabular form in Figure 5.9.

The set $\mathbf{S}_1$ contains three clauses:

$$\mathbf{S}_1 = \{\alpha_1, \alpha_2, \alpha_3\},$$

where

$$\begin{aligned}
\alpha_1 &= x_1{:}a \vee x_1{:}b \vee x_1{:}c \vee x_1{:}d, \\
\alpha_2 &= x_2{:}a \vee x_2{:}b \vee x_2{:}c \vee x_2{:}d, \\
\alpha_3 &= x_3{:}a \vee x_3{:}b \vee x_3{:}c \vee x_3{:}d.
\end{aligned}$$

The set $\mathbf{S}_2$ contains 18 clauses, six for each variable:

$$\mathbf{S}_2 = \{\beta_{rs}^{(i)} | 1 \leqslant i \leqslant 3, 1 \leqslant r, s \leqslant 4, r \neq s\},$$

where

$$\begin{aligned}
\beta_{1,2}^{(1)} &= \overline{x_1{:}a} \vee \overline{x_1{:}b}, & \beta_{1,3}^{(1)} &= \overline{x_1{:}a} \vee \overline{x_1{:}c}, \\
\beta_{1,4}^{(1)} &= \overline{x_1{:}a} \vee \overline{x_1{:}d}, & \beta_{2,3}^{(1)} &= \overline{x_1{:}b} \vee \overline{x_1{:}c}, \\
\beta_{2,4}^{(1)} &= \overline{x_1{:}b} \vee \overline{x_1{:}d}, & \beta_{3,4}^{(1)} &= \overline{x_1{:}c} \vee \overline{x_1{:}d}, \\
\beta_{1,2}^{(2)} &= \overline{x_2{:}a} \vee \overline{x_2{:}b}, \ldots, & \beta_{3,4}^{(2)} &= \overline{x_2{:}c} \vee \overline{x_1{:}d}, \\
\beta_{1,2}^{(3)} &= \overline{x_3{:}a} \vee \overline{x_3{:}b}, \ldots, & \beta_{3,4}^{(3)} &= \overline{x_3{:}c} \vee \overline{x_3{:}d}.
\end{aligned}$$

The set $\mathbf{S}_3$ contains the clauses representing the nogoods for the three relations R_1, R_2, and R_3:

$$\mathbf{S}_3 = \{\gamma_{rs}^{(h)} | 1 \leqslant h \leqslant 3, (a_{ir}, a_{js}) \notin R_h\}.$$

For relation $R_1(x_1, x_2)$ the set of nogoods is the following:

$$\text{nogoods}_1 = \{(a,a), (a,c), (b,a), (b,b), (b,c), (c,a), (d,b), (d,c), (d,d)\}.$$

Then

$$\begin{aligned}
\gamma_{1,1}^{(1)} &= \overline{x_1{:}a} \vee \overline{x_2{:}a}, \\
&\vdots \\
\gamma_{4,4}^{(1)} &= \overline{x_1{:}d} \vee \overline{x_2{:}d}.
\end{aligned}$$

In an analogous way we obtain

$$\begin{aligned}
\gamma_{1,1}^{(2)} &= \overline{x_1{:}a} \vee \overline{x_3{:}a},\\
&\vdots\\
\gamma_{4,4}^{(2)} &= \overline{x_1{:}d} \vee \overline{x_3{:}d},\\
\gamma_{1,2}^{(3)} &= \overline{x_2{:}a} \vee \overline{x_3{:}b},\\
&\vdots\\
\gamma_{4,4}^{(3)} &= \overline{x_2{:}d} \vee \overline{x_3{:}d}.
\end{aligned}$$

The propositional formula to be made true is thus

$$\begin{aligned}
\varphi(x_1{:}a, x_1{:}b, \ldots, x_3{:}d) = {} & (x_1{:}a \vee x_1{:}b \vee x_1{:}c \vee x_1{:}d) \wedge (x_2{:}a \vee x_2{:}b \vee x_2{:}c\\
& \vee x_2{:}d) \wedge (x_3{:}a \vee x_3{:}b \vee x_3{:}c \vee x_3{:}d) \wedge (\overline{x_1{:}a} \vee \overline{x_1{:}b})\\
& \wedge (\overline{x_1{:}a} \vee \overline{x_1{:}c}) \wedge \cdots \wedge (\overline{x_3{:}c} \vee \overline{x_3{:}d}) \wedge (\overline{x_1{:}a} \vee \overline{x_2{:}a})\\
& \wedge \cdots \wedge (\overline{x_2{:}d} \vee \overline{x_3{:}d}).
\end{aligned}$$

Number of clauses in $\mathbf{S}_1$

Let us now count how many clauses we have obtained with the preceding transformation. The set $\mathbf{S}_1$ contains as many clauses as there are variables; thus, $|\mathbf{S}_1| = n$. Each clause α_i $(1 \leqslant i \leqslant n)$ has as many terms as there are values in the domain D_i; hence $|\alpha_i| = d_i$.

Number of clauses in $\mathbf{S}_2$

For any variable x_i taking values in D_i, there are $\binom{d_i}{2}$ pairs of conflicting assignments. Thus

$$|\mathbf{S}_2| = \sum_{i=1}^{n} \binom{d_i}{2}. \tag{8.22}$$

Number of clauses in $\mathbf{S}_3$

Given a relation $R_h(x_i, x_j)$, the number of entries in the corresponding matrix $M_{ij}^{(h)}$ that take the value *false* is the difference between $|D_i \times D_j|$ and $|R_h(x_i, x_j)|$. Then, for each relation there are $(d_i d_j - |R_h(x_i, x_j)|)$ nogoods. The set $\mathbf{S}_3$ has thus cardinality

$$|\mathbf{S}_3| = \sum_{h=1}^{m} (d_i d_j - |R_h(x_i, x_j)|). \tag{8.23}$$

The total number of clauses is thus

$$\mathcal{T}(n, m, d_1, \ldots, d_n) = n + \sum_{i=1}^{n} \binom{d_i}{2} + \sum_{h=1}^{m} (d_i d_j - |R_h(x_i, x_j)|). \tag{8.24}$$

We may notice that the clauses in both $\mathbf{S}_2$ and $\mathbf{S}_3$ have two terms whereas the clause in $\mathbf{S}_1$, corresponding to the variable x_i, has d_i terms.

If the original CSP was generated by model B, then

$$|D_i| = d \qquad \forall i \in [1, n],$$
$$|R_h| = N \qquad \forall h \in [1, m].$$

Hence, the total number of clauses reduces to:

$$\mathcal{T}(n, m, d) = \mathcal{O}\left[n + n\binom{d}{2} + m(d^2 - N)\right].$$

The ratio of clauses and variables in the corresponding SAT will be, in this case,

Value of α in the corresponding SAT for a CSP generated with model B.

$$\alpha = \frac{1}{n}\left[n + n\binom{d}{2} + m(d^2 - N)\right] = 1 + \frac{d(d-1)}{2} + \frac{m}{n}(d^2 - N).$$

Using the expressions $p_1 = 2m/[n(n-1)]$ and $p_2 = 1 - N/d^2$, the control parameter α can be rewritten as

$$\alpha = 1 + \frac{d(d-1)}{2} + p_1 p_2 \frac{d^2(n-1)}{2}. \tag{8.25}$$

We now have a formal method for converting a CSP into a SAT problem. The immediate benefit is that we can use an algorithm for solving a SAT problem to solve a CSP.

8.5 Comments

Even if the transformations between propositional learning and SAT problems and between CSP and SAT are interesting *per se*, a question arises about their practical utility. One would like, on the one hand, to exploit the results found for SAT in order to predict when the converted problem is in the phase transition region and, on the other hand, to apply the very effective search algorithms designed for SAT. Unfortunately, both hopes remain to be realized. The classical models used for predicting the phase-transition location in SAT assume a fixed size k for all clauses, while the SAT instances obtained by transformation have clauses with a variable number of terms, leaving aside the fact that most results have been obtained only for $k \leqslant 3$. In addition, the utility of SAT solvers for learning and solving CSPs has not received a definite assessment; performances appear to depend on the specific problem approached.

9

Phase transition in FOL covering test

Contents

In this chapter we will discuss in depth the results that have emerged in recent work on the covering test in first-order logic (Giordana and Saitta, 2000; Maloberti and Sebag, 2004); this was introduced in Section 5.2.3, with particular emphasis on the DATALOG language. With this language, the covering test for a hypothesis $\varphi(x_1, \ldots, x_n)$ reduces to the problem of finding a substitution θ, for the variables $x_1, \ldots, x_n$, by a set of constants $a_1, \ldots, a_n$ that satisfies the formula φ.

Moreover, in Section 8.3 we showed how a covering test, i.e., the *matching problem* (φ, e), can be transformed into a CSP. As a consequence, a phase transition may be expected in the covering test, according to results obtained by several authors (Smith and Dyer, 1996; Prosser, 1996). As discussed in Chapter 4, studies on CSPs have exploited a variety of generative models designed to randomly sample areas of interest in the CSP space. As the CSPs occurring in

practice may have hundreds or thousands of variables and constraints, the investigations were oriented toward large problems; also, there was much interest in the asymptotic behavior of CSPs for large n.

Machine learning deals with small CSP problems.

In machine learning, even though the equivalence of the matching problem with a CSP suggests the emergence of a phase transition, the range of problem sizes involved is much smaller, as the typical number of variables and constraints in relational learning is rarely greater than 10. Thus, the first question we want to answer is whether a phase transition is present *in the region of the CSP space visited by a relational learner*. As uncovered by Botta *et al.* (1999) and by Giordana and Saitta (2000), a phase transition is indeed detected for problem sizes relevant to machine learning, even if it is less sharp than in the area explored by Prosser (1996). In this chapter we will discuss this in detail. A second relevant question concerns the *impact that the presence of a phase transition may have upon the quality, or even the very feasibility, of relational learning*. Chapter 10 answers this question.

Investigating the covering test in first-order logic, Giordana and Saitta (2000) implicitly proposed a generative model of matching problems of the type encountered in machine learning. Thus our starting point will be to revisit this model, which we will name model RL,[1] making explicit its properties and relationship to the classical models. Then we will review and discuss the experiments of Giordana and Saitta (2000) and of Maloberti and Sebag (2004). Finally, we will compare the findings obtained using model RL with those obtained by other authors.

9.1 Model RL

Generative models used for investigating CSPs are characterized by:

- a set of control parameters describing the CSP space;
- a set of basic assumptions restricting the control parameters;
- an algorithm for the stochastic generation of CSP instances.

Model RL was designed to investigate the occurrence of a phase transition in the matching problem in relational learning.

An aspect that differentiates model RL from other models is the choice of control parameters. Instead of using the classical p_1 and p_2 directly (see Section 4.2), four different parameters, two characterizing the example complexity and two characterizing the hypothesis complexity, are chosen, as described in the following. The generative algorithm is also different, so that problem instances preserve some plausibility with respect to real learning problems.

[1] The notation RL indicates that the model was designed to investigate CSPs in relational learning.

9.1.1 The control parameters

Before entering into the discussion of control parameter selection, we recall that a matching problem is a pair (φ, e), where φ is a hypothesis (a DATALOG formula) and e is an example (a set of tables). The example e constitutes the universe U, where φ must be proved *true* or *false* by performing the covering test.

Control parameters characterizing learning examples
In relational learning an example e is defined by a set of atomic objects described by a set of attributes and a set of relations stating their mutual interactions. The atomic objects occurring in the universe U corresponding to e are given and, in a logical framework, they are identified by means of a set Λ of L constants (symbols) a_i:

$$\Lambda = \{a_1, a_2, \ldots, a_L\}. \tag{9.1}$$

The first parameter, which naturally appears as a good candidate for characterizing the complexity of an example, is the number L of its atonic components, i.e., the cardinality of the set Λ.

As discussed in Section 5.2.2, relations between objects are represented by means of tables, as is common in relational databases. Every instance of a relation R_h involving a group of objects is encoded by a row in the corresponding table. Let N_h denote the cardinality of the relation R_h (i.e., the number of rows in the corresponding table) and r_h denote its arity (i.e., the number of columns in the associated table). It is immediate to see that R_h is a subset of the maximal table Λ^{r_h} obtained as the Cartesian product of set Λ with itself r_h times. If the tuples that populate the table are sampled from Λ^{r_h} uniformly and with replacement, the probability that a tuple $\langle a_1, a_2, \ldots, a_{r_h}\rangle$ is included at least once in R_h is given by

$$\mathtt{P}_h = \mathtt{P}(\langle a_1, a_2, \ldots, a_{r_h}\rangle \in R_h) = 1 - \left(1 - \frac{1}{L^{r_h}}\right)^{N_h}, \tag{9.2}$$

where N_h is the cardinality of R_h. Alternatively, if sampling is performed without replacement the probability is given by

$$\mathtt{P}_h(\langle a_1, a_2, \ldots, a_{r_h}\rangle \in R_h) = \frac{N_h}{L^{r_h}}. \tag{9.3}$$

We note that (9.2) and (9.3) tend to coincide when $L \to \infty$. In fact, for $x \to 0$ we have $(1-x)^t \backsimeq 1 - tx$.

It is of interest to observe how L and N_h interact. From equations (9.2) and (9.3) we see that, in both cases, increasing N_h means that $\mathtt{P}_h$ increases whereas increasing L_h means that $\mathtt{P}_h$ decreases. Thus if R_h is a binary constraint in a CSP ($r_h = 2$), the probability of obtaining a solution is directly correlated with

N_h and inversely correlated with L^2. Therefore we will use the N_h and L as parameters to characterize the complexity of e.

Control parameters depending on the hypothesis description
Let $\varphi(x_1, x_2, \ldots, x_n)$ be the hypothesis to be verified in e. Both the complexity of the covering test and the probability of success depend on two features of φ: the number n of variables and the number m of literals (i.e., predicates or their negation) occurring in φ. We recall that in DATALOG the covering test corresponds to the problem of finding a substitution

$$\theta = (x_1/a_1, x_2/a_2, \ldots, x_n/a_n), \tag{9.4}$$

for variables $x_1, x_2, \ldots, x_n$ with constants a_i from the domain Λ, such that $\varphi\theta$ is satisfied. The parameter n defines the tuple length and characterizes the size of the space Θ from which θ must be selected. More specifically, the size of Θ is $|\Theta| = L^n$.

The parameter m defines the number of constraints occurring in φ. We remember that each predicate p_h occurring in φ is associated with a specific relation R_h defined in e. Thus p_h denotes a constraint defined by R_h, which rules out a subset of the possible substitutions Θ.

Note that keeping constant the number of variables n and increasing the number of literals m in a hypothesis φ reduces the number of substitutions $\theta \in \Theta$ that satisfy φ, making φ more difficult to satisfy. Conversely, increasing n while m remains constant increases the number of substitutions θ satisfying φ, making φ easier to satisfy. In our context, we select the number of predicates m as a second control parameter.

9.1.2 Model assumptions

In order to obtain a manageable form of the matching problem (φ, e), while preserving the essence of the problem itself, we need to make some assumptions about the control parameters to be used and the structure of the hypothesis and of the example.

The parameters n (the number of variables) and m (the number of literals) have an immediate interpretation in φ and can be used as control parameters. However, the cases of L and N_h need some discussion. Every relation R_h occurring in e may have, in principle, a different size N_h. Moreover, in the real world every relation R_h can be selected from the space Λ^{r_h} using a different criterion, i.e., a different distribution, depending on the specific application. In addition, not all the L constants may appear in all relations. Thus it is not possible to obtain N_h and L from an example without making further assumptions.

Assumption 1 *All relations have the same size N, the same arity r and the same distribution over the space Λ^r.*

All relations have the same size.

The choice in Assumption 1 focuses the attention on specific classes of problems that are relevant to practical learning applications while potentially containing very hard instances (Giordana and Saitta, 2000). This class is characterized by examples containing large numbers of atomic objects of only a few different types, linked by the same type of relations. Problems of this class are frequently found, for instance, in chemistry and molecular biology.

Assumption 1 allows the matching problem (φ, e) generated by the model RL to be characterized by a 4-tuple $\langle n, m, L, N \rangle$.

Assumption 2 *Target relations have 0-arity and conjunctive descriptions.*

Target concepts are 0-arity relations with conjunctive descriptions.

As discussed in Chapter 5, the simplest case of relational learning consists in learning 0-arity relations (constants) corresponding to concept descriptions of the form:

$$\varphi(x_1, x_2, \ldots, x_n) \rightarrow c, \tag{9.5}$$

where c is a constant (a concept name), which is *true* or *false* in a given example e depending on whether φ is *true* or *false* in e. We also remember that variables in a clause not occurring in the head (in this case the right-hand side of the implication) are implicitly existentially quantified. Moreover, the formula φ describing the concept is assumed to be a conjunction of literals. Thus in this case the covering test reduces to the verification of an existentially quantified formula of the type $\exists \vec{\mathbf{x}}\ [\varphi(\vec{\mathbf{x}})]$, where $\varphi(\vec{\mathbf{x}})$ is a conjunction of literals.

As a matter of fact assumption 2 is not strongly limitative, because in supervised learning the complexity of the covering test depends only on the complexity of verifying the existentially quantified part of the formula. In fact, the possible bindings for the variables occurring in the head of a hypothesis (the target relation) are fixed by the *a priori* knowledge given by the teacher. Apart from its simplification of the matching problem, this assumption also allows us to compare, in the next chapter, various relational learners independently of their ability to learn relations having arity greater than 0.

Assumption 3 *Only binary relations are considered in the examples and, consequently, only binary predicates will occur in the hypotheses.*

This third assumption ($r = 2$) greatly simplifies the model while preserving its plausibility. In addition, it allows our results to be compared directly with the

many available in the literature for binary CSPs. In principle, relations of any arity may occur in a universe e, including unary relations and relations of arity greater than 2. Nevertheless, a restriction to binary relations is not limitative for two reasons. The first is that most existing relational learners work with unary and binary relations only. The second is that it is always possible to transform a universe containing relations of different arity into a universe containing only binary relations.

Concept descriptions are built on binary relations.

Assumption 4 *Only connected formulas are considered.*

This fourth assumption concerns the structure of the hypothesis. We say that a formula φ is *unconnected* when there exist two (or more) subsets of variables $\vec{\mathbf{x}}_1 = \{x_1, x_2, \ldots, x_k\}$ and $\vec{\mathbf{x}}_2 = \{x'_1, x'_2, \ldots, x'_r\}$ such that no literal in $\varphi(\vec{\mathbf{x}}_1, \vec{\mathbf{x}}_2)$ has variables in both $\vec{\mathbf{x}}_1$ and $\vec{\mathbf{x}}_2$. In fact, if two such subsets exist then a hypothesis, under assumption 1, can always be reduced to the conjunction of two independent existentially quantified subformulas:

$$\exists\, \vec{\mathbf{x}}_1,\ \vec{\mathbf{x}}_2\ [\varphi(\vec{\mathbf{x}}_1, \vec{\mathbf{x}}_2)] \equiv \exists \vec{\mathbf{x}}_1 [\psi(\vec{\mathbf{x}}_1)] \wedge \exists \vec{\mathbf{x}}_2 [\rho(\vec{\mathbf{x}}_2)]. \tag{9.6}$$

In this case, the original problem is reduced to two simpler problems, consisting in matching $\exists \vec{\mathbf{x}}_1 [\psi(\vec{\mathbf{x}}_1)]$ and $\exists \vec{\mathbf{x}}_2 [\rho(\vec{\mathbf{x}}_2)]$ separately on e.

In order to avoid this case, model RL is prevented from generating unconnected hypotheses. This restriction is not limitative. On the one hand, most existing relational learners will make assumption 4 and so will not generate unconnected hypotheses. In fact, allowing a relational learner to construct such hypotheses entails a tremendous increase in the size of the region of hypothesis space visited by the learner. On the other hand, the complexity of the covering test for unconnected formulas can be simply obtained as the sum of the complexity of the unconnected components. Thus investigating the covering test complexity for connected formulas is sufficient.

Only connected formulas are considered.

Assumption 5 *Learning examples contain only relations corresponding to the literals occurring in φ.*

This last assumption concerns the structure of the examples. More specifically, in model RL it is assumed that a positive example only contains relations that occur in the target concept, namely in examples where there are no *irrelevant* predicates. This assumption is a simplification that does not affect the complexity of the covering test; irrelevant relations, i.e., relations that do not correspond to any literal in the formula to be verified, are simply disregarded by any theorem prover. On the contrary, it makes significantly easier the inductive search of the

All relations in an example are relevant to the concept description.

learner, as we will discuss in the following chapter. In fact, a large number of irrelevant hypotheses are implicitly ruled out.

Therefore, under the above five assumptions, every pair (φ, e) generated by model RL will correspond to a specific point in the space $\langle n, m, N, L\rangle$.

9.1.3 Matching problem generation

We are now ready to describe the algorithm used to construct CSPs belonging to the space characterized by model RL. The input of the algorithm is a 4-tuple $\langle n, m, N, L\rangle$ chosen *a priori*. The CSP construction occurs in two steps. The first generates the universe e (the example), while the second constructs the hypothesis φ to be verified in e. More specifically, the second step first constructs a "skeleton" of the formula, thus guaranteeing that it is connected; then, other literals are added until the required number m of literals is reached.

n variables
m literals
L constants
N rows in relational tables

Let $\mathbf{X} = \{x_1, x_2, \ldots, x_n\}$ denote the set of n variables, and $\mathbf{P} = \{p_1, \ldots, p_m\}$ the set of m literals.

Given L and N, e is fully defined by a set $\mathbf{R}$ of m relational tables, each one associated with a predicate in the set $\mathbf{P}$. According to the assumptions made, all tables are binary and have the same size N. Every table is populated by sampling at random, without replacement, N pairs from the set $\Lambda \times \Lambda$. The table construction proceeds as in step 1 below.

An example is a universe containing tables corresponding to the predicates defined in the hypothesis description language.

Step 1 *Relational table construction* Let R denote the table, initially empty, to be populated. Let s be the current number of tuples added to R.

1. Construct the universal binary relation $\Lambda \times \Lambda$, and set $s = 0$.
2. While $s < N$, extract uniformly, without replacement, one pair t from $\Lambda \times \Lambda$ and add t to R. Increment s by 1.
3. Add R to example e.

Notice that, as pairs are sampled without replacement, no relation will contain duplicate tuples.

Step 2 *Construction of hypothesis* φ Starting from $\mathbf{X}$ and $\mathbf{P}$, and assuming that $m \geq n - 1$, a formula $\varphi(\vec{\mathbf{x}})$ is constructed as follows:

1. Set $\varphi(\vec{\mathbf{x}}) = \top$, the most general hypothesis.
2. Form the sequence of variables $x_1, x_2, \ldots, x_n$.

3. **For each** pair of consecutive variables (x_i, x_{i+1}) $(1 \leqslant i \leqslant n-1)$ **do**
   - Extract uniformly, without replacement, a predicate from $\mathbf{P}$ and apply it to the variable pair (x_i, x_{i+1}).
   - Call $\alpha_i(x_i, x_{i+1})$ the literal so obtained and add it to the current formula as a conjunct.
4. **For each** of the remaining $m - n + 1$ predicates in $\mathbf{P}$, **do**
   - Extract a pair of variables (y_j, z_j) from $\mathbf{X} \times \mathbf{X}$ such that $y_i \neq z_i$, call $\beta_j(y_j, z_j)$ the literal so obtained, and add it to the current formula as a conjunct.

In step 2, the third point constructs the "skeleton" of the connected formula and guarantees that all n variables are *chained*, so that the matching problem cannot be reduced to smaller problems (with smaller n values). The fourth point brings the number of predicates in $\varphi(\vec{\mathbf{x}})$ to the desired value m. Note that no predicate is negated. The formula generated contains exactly n variables and m predicates, and the same pair of variables may appear in more than one predicate. The final formula assumes the following format:

$$\varphi(\vec{\mathbf{x}}) = \bigwedge_{i=1}^{n-1} \alpha_i(x_i, x_{i+1}) \wedge \bigwedge_{j=1}^{m-n+1} \beta_j(y_j, z_j). \tag{9.7}$$

The above generation procedure is close to *model B*, introduced by Smith and Dyer (1996) for CSPs. Similarities and differences will be discussed later in this chapter. The hypothesis space $\mathcal{H}$ to be explored contains a number $|\mathcal{H}_{RL}|$ of syntactically different formulas:

The procedure for generating the example is similar, but not identical, to that used in model B.

$$|\mathcal{H}_{RL}| = \binom{m}{n-1}(n-1)!(n^2-n)^{m-n+1}. \tag{9.8}$$

Expression (9.8) is computed as follows: when constructing the first part of formula (9.7), $n-1$ predicates are selected without replacement from a set of m; this selection gives $\binom{m}{n-1}$ alternatives. Each of the $n-1$ selected predicates can be applied to any pair (x_i, x_{i+1}) of variables; each permutation of the assignments generates a different formula, because the predicates are all different, as well as the variable pairs. Thus there are $\binom{m}{n-1}$ $(n-1)!$ alternatives for the first part of formula (9.7). Regarding the second part of the formula, the predicates are the $m - n + 1$ not yet used; each can be applied to any combination of different variable pairs. The total number of variable pairs is n^2, but we have to subtract pairs where the same variable occurs in both places, i.e.,

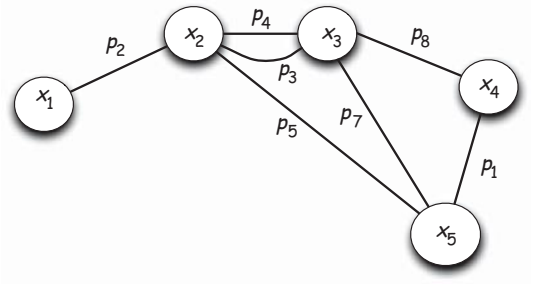


Figure 9.1 Example of a constraint graph corresponding to a formula generated by model RL, starting from a set $\mathbf{X} = \{x_1, x_2, x_3, x_4, x_5\}$ of five variables and a set $\mathbf{P} = \{p_1(x,y),\ p_2(x,y), p_3(x,y), p_4(x,y), p_5(x,y), p_6(x,y), p_7(x,y), p_8(x,y)\}$ of eight predicates. The formula is $\varphi(x_1, \ldots, x_5) = \big[p_2(x_1,x_2) \wedge p_4(x_2,x_3) \wedge p_8(x_3,x_4) \wedge p_1(x_4,x_5)\big] \wedge \big[p_3(x_2,x_3) \wedge p_5(x_2,x_5) \wedge p_7(x_3,x_5)\big]$.

n pairs. Thus the number of different alternatives for the third factor in (9.7) is $(n^2 - n)^{m-n+1}$.

In Figure 9.1 an example of a constraint graph corresponding to a formula generated by model RL is given.

We can also count the number of different examples that can be constructed by model RL:

$$|\mathcal{E}_{RL}| = \binom{L^2}{N}^m \tag{9.9}$$

Globally, model RL can generate a number of matching problems equal to

$$|\mathcal{H}_{RL}|\ |\mathcal{E}_{RL}| = \frac{m!\ (n^2-n)^{m-n+1}}{(m-n+1)!} \binom{L^2}{N}^m .$$

Notice that, according to assumption 3, any φ generated in this way contains binary predicates only. Moreover, according to assumption 5, e contains all and only the relations corresponding to the literals occurring in φ. The natural logarithm of $|\mathcal{H}_{RL}|$ is plotted in Figure 9.2 as a function of n and m.

EXAMPLE

Let us consider the variable set $\mathbf{X} = \{x_1, x_2, x_3\}$ and the predicate set $\mathbf{P} = \{on(x,y), left(x,y), adjacent(x,y)\}$. In this case $n = 3$ and $m = 3$. The predicate $on(x,y)$ states that object x touches object y from above and the predicate $left(x,y)$ states that object x is to the left of object y whereas the predicate $adjacent(x,y)$ states that objects x and y touch each

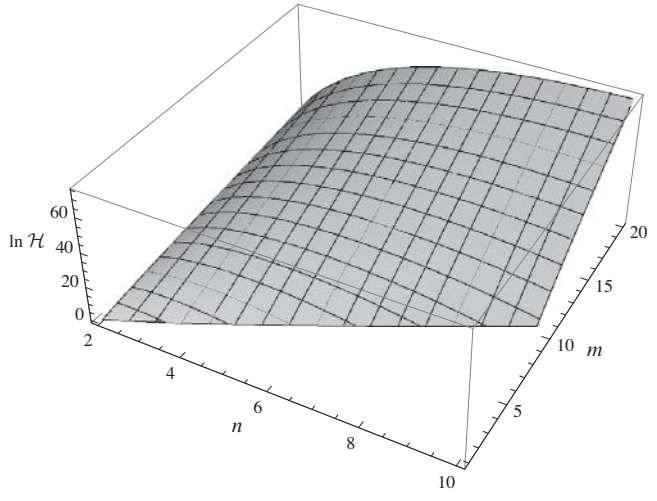


Figure 9.2 Natural logarithm of the function $\ln\ |\mathcal{H}_{RL}|$ vs. n and m.

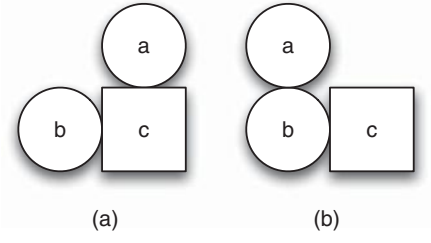


Figure 9.3 (a) An instance satisfying the formula $\exists\vec{\mathbf{x}}[\varphi_1(\vec{\mathbf{x}})]$. (b) An instance satisfying the formula $\exists\vec{\mathbf{x}}[\varphi_2(\vec{\mathbf{x}})]$.

other laterally. The predicate $adjacent(x, y)$ is symmetrical. Two possible formulas are

$$\exists\vec{\mathbf{x}}[\varphi_1(\vec{\mathbf{x}})] = left(x_1, x_2) \wedge on(x_2, x_3) \wedge adjacent(x_3, x_1),$$
$$\exists\vec{\mathbf{x}}[\varphi_2(\vec{\mathbf{x}})] = on(x_1, x_2) \wedge adjacent(x_2, x_3) \wedge left(x_1, x_3).$$

In Figure 9.3 one instance of each formula is provided.

An interesting point to be discussed is the structure of the constraint graph $\boldsymbol{G}$. In fact, owing to the fact that predicates may share the same pair of variables, the number s of edges in $\boldsymbol{G}$ is a stochastic variable whose value depends upon m

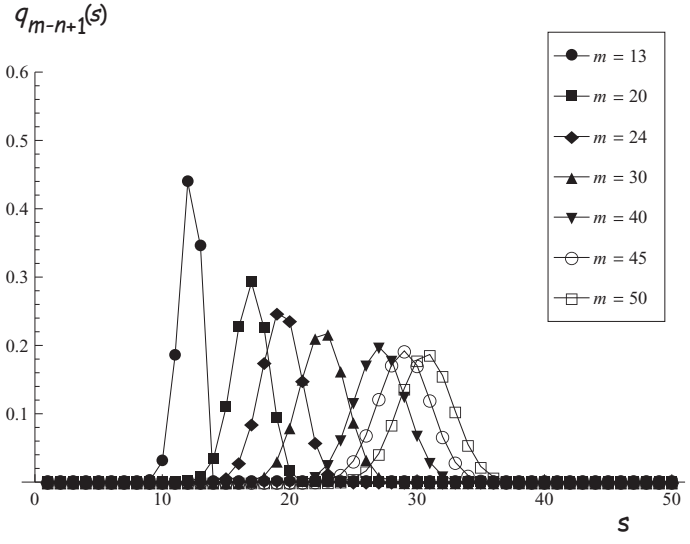


Figure 9.4 Probability distribution $q_{m-n+1}(s)$ as a function of the number of edges s in the constraint graph. The curves are shown for $n = 10$ and various values of m. The curves start at $s = n - 1 = 9$. (Note that the defining symbols merge along the s axis.)

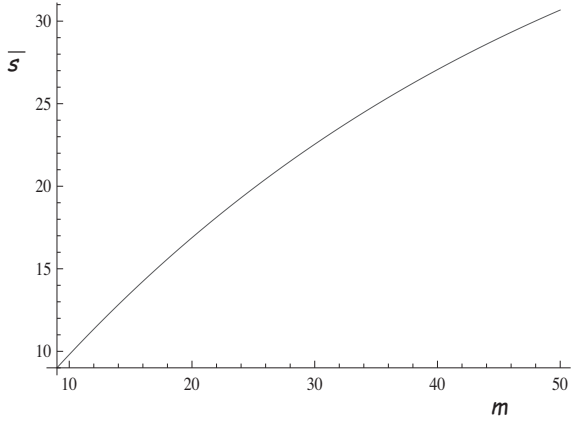


Figure 9.5 Average value $\bar{s}$ of the number of edges in the constraint graph generated by model RL, as a function of the number m of predicates.

and n. All the graphs that can possibly be generated belong to an ensemble $\mathcal{G}_{RL}$, which can be partitioned according to the number of edges:

$$\mathcal{G}_{RL} = \bigcup_{s\,=\,n-1}^{m} \mathcal{G}_s.$$

All graphs belonging to a particular $\mathcal{G}_s$ have the same probability of occurrence, but the probability that a constraint graph belongs to $\mathcal{G}_s$ is not uniform over the number of edges s. In order to compute this probability, let us consider what happens in point 4 of step 2 of the above generation algorithm. In the following, let us define $M = n(n-1)/2$. A number $m - n + 1$ of extractions of pairs of (different) variables is performed in sequence and with replacement. Then, let us define by $q_k(s)$ $(n-1 \leqslant s \leqslant \min[n+k-1, M])$ the probability distribution of s *after* the kth extraction. At each extraction the number s may either remain the same (if the extracted pair already appears in the formula) or may increase by 1 (if the extracted pair does not already appear in the formula). As extraction is uniform, the probability that an already existing pair is extracted is proportional to the current number of existing pairs. We can provide a recursive definition of the probability $q_k(s)$:

$$q_0(s) = \delta(s, n-1),$$

$$q_k(s) = \begin{cases} 0 & \text{if } s \leqslant n-2, \\ [(n-1)/M]^k & \text{if } s = n-1, \\ q_{k-1}(s-1)\,[1-(s-1)/M] & \\ \quad + q_{k-1}(s)s/M & \text{if } n \leqslant s \leqslant \min[M, n+k-1], \\ 0 & \text{if } s \geqslant \min[M, n+k-1]+1. \end{cases}$$

It is easy to prove, using induction over k, that $q_k(s)$ is indeed a probability distribution, as it is normalized for each $k \in [0, m-n+1]$ over the values $s \in [n-1, \min[M, n+k-1]]$.

Notice that, from the point of view of the graph structure, the pair (x_i, x_j) is equivalent to the pair (x_j, x_i). The final probability distribution is that obtained for $k = m-n+1$. In Figure 9.4 the probability distributions obtained for $n = 10$ and $m = 13, 20, 24, 30, 40, 45, 50$ are given. In Figure 9.5 the average value of s is shown for $10 \leqslant m \leqslant 50$.

9.2 The search algorithm

Before addressing the experimental analysis of the covering test complexity for formulas generated by model RL, we need to select the search algorithm. Given

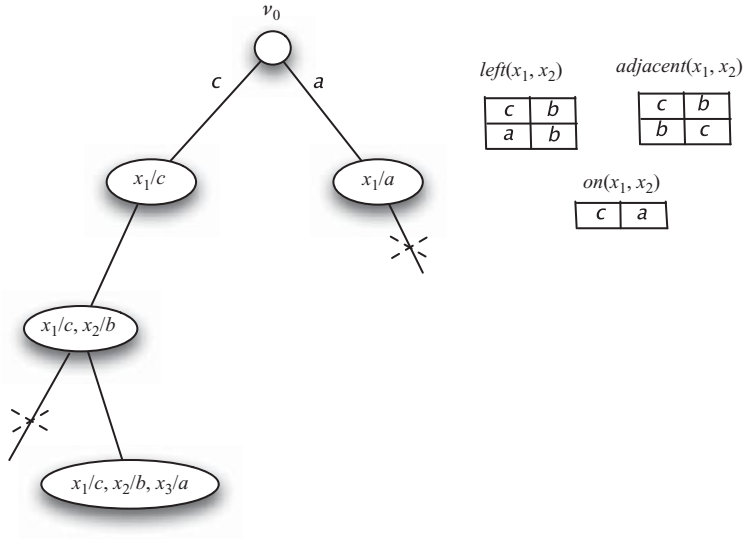


Figure 9.6 The search tree explored by the covering test for the formula $\exists\vec{\mathbf{x}}\,[\varphi(\vec{\mathbf{x}})] = on(x_1, x_3) \wedge left(x_1, x_2)$.

a formula φ, with n variables and the syntactic structure (9.7), and a universe e, the search for a substitution θ satisfying φ in e entails visiting a tree τ, as indicated in Figure 9.6. A node ν at a level k in τ corresponds to an allowed substitution θ for the variables $x_1, \ldots, x_k$, considered in a given sequence.[2] The leaves of τ at the 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 complexities. However, the primary goal of the investigation reported in the following was not to reduce the search complexity but to design a generic (unbiased) algorithm to be used as a standard tool to measure and compare complexity. Smart, sophisticated, search algorithms tend to exhibit performances that are dependent on the specific problem configuration.

The stochastic search algorithm that we propose guarantees performances more homogeneous than those of deterministic backtracking algorithms.

A natural reference baseline is represented by backtrack search algorithms, which have been the starting point for most problem-solving methods in artificial intelligence and are still now at the heart of the inference mechanism in logic programming environments such as Prolog. However, backtrack algorithms visit

[2] Different orderings of the variables, both static and dynamic, have been tried with no noticeable change in the emergence of the phase transition.

the search tree in a preassigned order and exhibit very different performances depending on the position of the solutions in the tree. An alternative is represented by stochastic search algorithms, which show more homogeneous performances. A comparison between a backtrack deterministic search algorithm and a stochastic search algorithm, in the context of the present study, was presented by Botta *et al.* (1999). Based on that comparison, the analysis reported here was done using a stochastic algorithm because it offers the advantage of both a lower average complexity and a lower complexity variance than a deterministic algorithm.

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 $MC(\tau, n)$ be a Monte Carlo stochastic algorithm, i.e., an algorithm that, according to the classification of Brassard and Bratley (1988), always provides an answer; however, the answer may be incorrect. The parameters τ and n of the function denote the search tree and the number of variables (the maximum depth of the tree), respectively. The algorithm $MC(\tau, n)$ explores one path on the search tree (see the example in Figure 9.6), starting from the root ν_0 and ending in a leaf ν, which may or may not be a solution. During the algorithm's execution, ν is associated with a sequence of nodes in the tree at increasing depth, and corresponds to increasingly complete, allowable, partial assignments of values to the variables $x_1, \ldots, x_n$. By iterating MC on τ as follows, more and more paths are explored:

$MC(\tau, n)$
$\nu = \nu_0,\ leaf\ =\ false$
while$(\neg leaf)$ **do**[3]
  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 $\nu'\ from\ C(\nu)$ with uniform probability
      $Set\ \nu = \nu'$
  endif
end
Label ν is closed
if the level of ν is $k = n$ **then** answer YES **else** $answer\ NO$.

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

[3]The notation $\neg leaf$ means the negation of $leaf$.

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 is chosen, the *Selectable* predicate will not include in $C(\nu)$ any node that either is closed (already explored) 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 that of an exhaustive backtrack search algorithm. The experiments reported in this chapter were done using search without replacement.

9.3 Experimental analysis

In order to locate the phase transition, points in the (m, L) plane were systematically explored for fixed values of the parameters n and N. More specifically, for $n = 4, 6, 10, 12, 14$ and for $N = 50, 80, 100, 130$ the complete mesh, covering the region $(10 \leq L \leq 50, n - 1 \leq m \leq 50)$ in the plane (m, L), was considered. For each pair (m, L) belonging to the mesh, 100 problems were generated, giving a total of about 900 000 problems. The range of n was chosen to be consistent with that employed in relational learning, where only a few variables have been considered so far.

9.3.1 Probability of solution

A three-dimensional plot representing the empirical probability of solution P_{sol} as a function of m and L is shown in Figure 9.7 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 at that point.

A sharp transition from a YES region to a NO region appears.

The graph in Figure 9.7 is very steep. To the left of the descent it shows a plateau, where the probability of finding a solution is almost equal to 1 (all the generated matching problems were solvable); we call this the YES *region*. To the right the graph again shows a plateau, where the probability of finding a solution is almost equal to 0 (no generated matching problem was solvable); we call this the NO *region*. In between, where the graph values rapidly drop from almost 1 to almost 0, is the *phase transition* region, also called the *mushy region* (Smith, 1994). The ideal phase transition location coincides with the set of points on the graph where $P_{sol} = 0.5$.

An interesting feature of the graph is the regularity of the projection onto the (m, L) plane of the contour level plot at $P_{sol} = 0.5$; this projection is a very smooth curve with a hyperbolic-like behavior. Figure 9.8(a) shows these

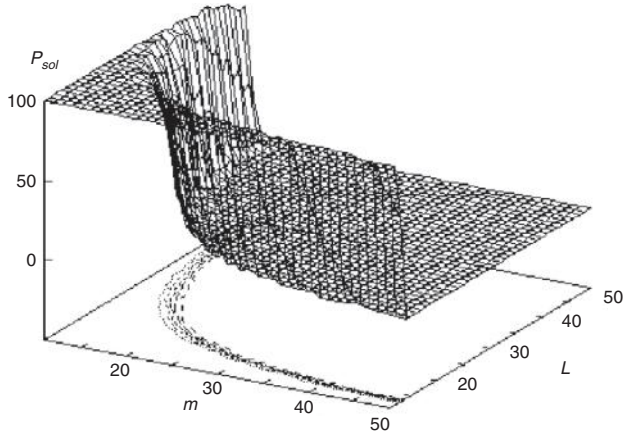


Figure 9.7 Three-dimensional plot of the probability of solution P_{sol} for $n = 10$ and $N = 100$. Some contour level plots, corresponding to P_{sol} values in the range $[0.85 \div 0.15]$,[4] have been projected onto the (m, L) plane.

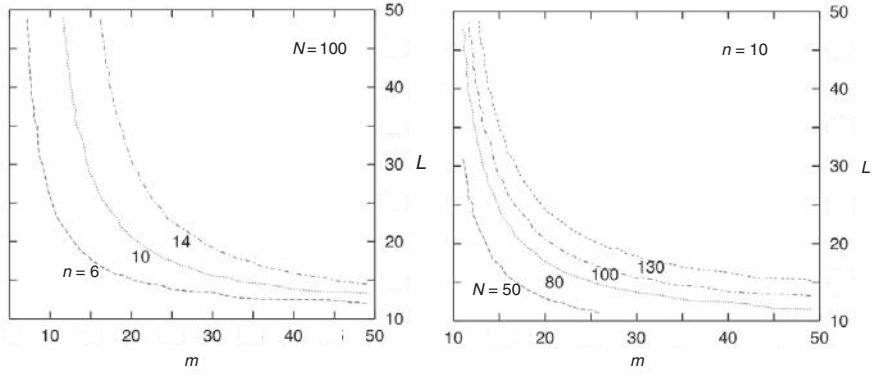


Figure 9.8 Plots of the 0.5-level contour of the probability of solution P_{sol}. (a) Graphs corresponding to the numbers of variables $n = 6$, 10, and 14, with relation cardinality $N = 100$. As we must have $m \geqslant n - 1$ for each m value, the highest acceptable graph is that for $n = m + 1$. (b) Graphs corresponding to relation cardinalities $N = 50$, 80, 100, and 130, with $n = 10$. As we must have $N \leqslant L^2$ for each value of L, the rightmost acceptable graph is that for $N = L^2$.

[4] This can also be written as $[0.85, 0.15]$.

projections for numbers of variables $n = 6, 10$, and 14, with relation cardinalities $N = 100$. Figure 9.8(b) shows an analogous set of contour plots for a constant number of variables $n = 10$ and for relation cardinalities $N = 50, 80, 100$, and 130.

9.3.2 Search complexity

For a quantitative analysis of the complexity, a random search without replacement was performed by repetition of the Monte Carlo algorithm described in Section 9.2. The cost $\mathcal{C}$ of the search was defined as the total number of nodes explored 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 9.9(a) the graph of the search complexity $\mathcal{C}$, averaged over 100 repetitions for each point, is shown, for $n = 10$ and $N = 100$. The shape and location of the highest-complexity region roughly matches the transition in probability seen in Figure 9.7 but it is more irregular and also broader, like a mountain chain. Within the "mountain" there is a large variability between different problems, witnessed by the variance plot shown in Figure 9.9(b). As one might expect, the highest-variance values correspond to the highest peaks. The maximum-complexity contour coincides with the contour plot at $P_{sol} = 0.5$, as found previously by, for instance, Hogg (1996), Hogg *et al.* (1996), and Cheeseman *et al.* (1991).

A complexity peak appears in correspondence with the phase transition.

It is worth noticing that the complexity distributions for solvable and unsolvable problems may be very different, as unsolvable problems usually require much more search. Approximations to the complexity probability distributions at the phase transition for solvable and unsolvable CSPs were provided by Frost *et al.* (1997). They showed that a lognormal distribution is a good approximation for unsolvable problems. For solvable problems several known distributions (in particular, the Weibull distribution) were tried with less success. However, from all the reported experiments it clearly emerges that the complexity distributions of both solvable and unsolvable problems have a long tail in the region of extremely hard problem instances. Finally, Figure 9.9(c) shows the dependency of the CPU time upon the number of nodes explored, 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. Conversely, implementation S_2 makes use of implicit indexing 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 $\mathcal{C}$. Implementation S_2 is a reasonable

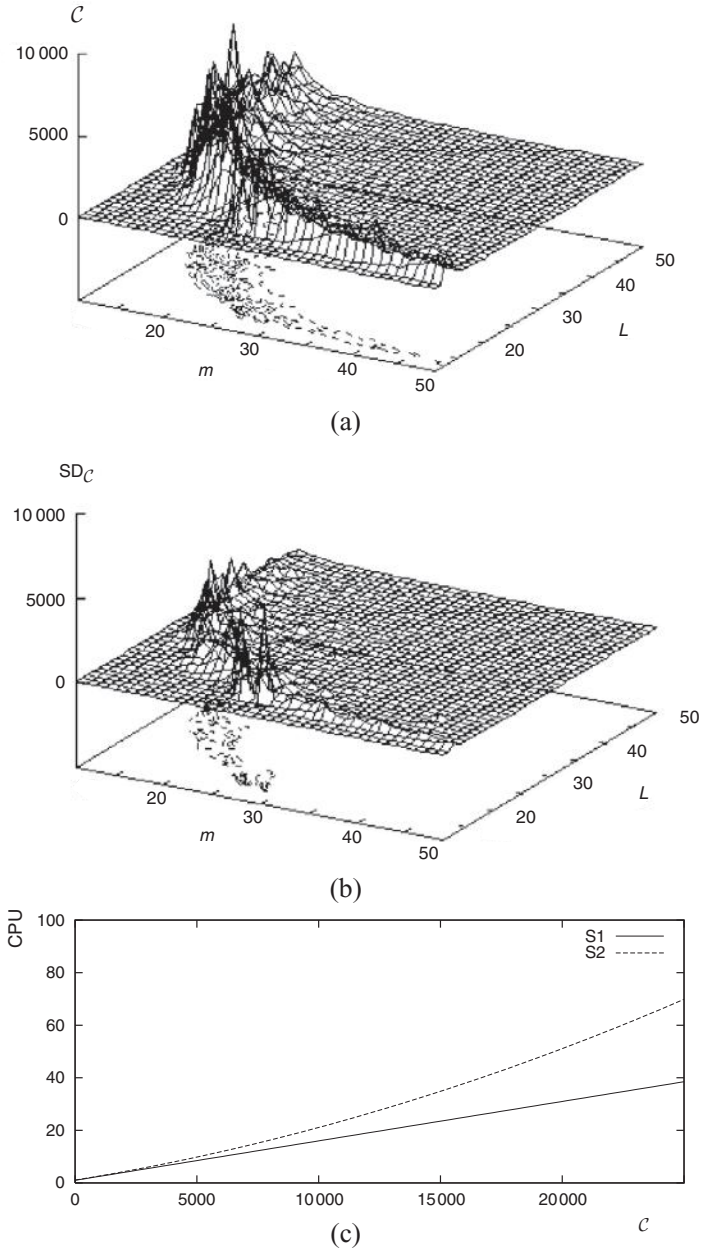


Figure 9.9 (a) Plot of the complexity $\mathcal{C}$ of the Monte Carlo stochastic search algorithm MC without replacement, for $n = 10$ and $N = 100$. Each point is the average over 100 problem instances. (b) Plot of the standard deviation of the complexity. (c) CPU time in centiseconds vs. the complexity of the search for two different implementations, S_1 and S_2, of the stochastic matching algorithm.

trade-off and was used for most experiments reported here.[5] The measures were obtained using a Pentium II-366.

9.4 Comparing model RL with other models for CSP generation

We will now provide an interpretation of the experimental findings given in the previous section for model RL and compare them with other models discussed in the literature (see Section 4.2). Figure 9.8 shows the hyperbolic-like shape of the phase transition edge in the plane (m, L), which moves far from the origin as parameters n and N increase.

9.4.1 Explaining the findings with model RL

Increasing L for constant n and N increases the constraint tightness.

An intuitive explanation of the observed hyperbolic behavior is obtained by considering how the probability value P_α that a substitution $\{x_i/a_i, x_j/a_j\}$ satisfies a generic relation $\alpha(x_i, x_j)$ depends upon L (see (9.1)). According to (9.3) the probability P_α that the pair (a_i, a_j) occurs in the table R_α is N/L^2, as sampling is done without replacement in model RL. Thus the effect of increasing L while N remains constant corresponds to a decrease in the value P_α, i.e., an increase in hardness of the constraint set for each literal occurring in a formula. This explains why the phase transition edge moves closer and closer to the L axis as L increases. In fact, if we observe that the larger is m the harder it is to satisfy the formula (for N constant) then increasing L means that even very small formulas (with only a few predicates) become unsatisfiable. The opposite is also true: if L decreases, it is necessary to construct very long formulas (large m values) for such formulas to be unsatisfiable. However, in this case L cannot decrease down to 0. In fact, when $L = \sqrt{N}$, the probability P_α that the pair (a_i, a_j) occurs in R_α is 1 because, for this value of L, all pairs in Λ^2 occur in the table and the formula φ is satisfied for any value of m. Thus the horizontal line $L = \sqrt{N}$ is a limiting curve in the plane (m, L). Another limiting line is the vertical at $m = n - 1$.

Let us now consider Figure 9.8(b) and the effect that an increase in the cardinality N of the relations has on the location of the phase transition. For constant

[5] During the generation of the graph in Figure 9.9(a) (involving 160.000 matching problems), S_2 never exceeded the memory size of 2 Mbytes whereas S_1 grew to 12 Mbytes. The time elapsed was about 215 min for S_1 and 280 min for S_2. When formulas with 14 variables were being processed, several times S_1 was unable to finish whereas S_2 exhibited a typical size of 2 Mbytes in the phase transition region and reached a maximum of 56 Mbytes. The time elapsed was 1650 min for S_2 whereas S_1 ran for several days due to its intensive use of the virtual memory.

L, increasing N means increasing P_α. Let us consider in Figure 9.8(b) a particular curve corresponding to a value N_1, and let $N_2 > N_1$. For each horizontal line corresponding to a specific value of L, the intersection of the phase transition line for N_2 with this horizontal line will be located *to the right* of the intersection point with the phase-transition line corresponding to N_1. In fact, increasing N will extend the YES region, which is located to the left of the phase transition boundary. As a consequence the whole phase transition line will be displaced towards the right as N increases. However, for any finite N, $P_\alpha \to 0$ for $L \to \infty$.

In order to understand how the phase transition boundary depends on the number n of variables (see Figures 9.8(a) and 9.10), we need to consider how the probability $P_{sol} = P_\varphi$ that a formula φ is satisfied depends upon the structure of φ. In fact, increasing or decreasing n affects the distribution of the literals over the variables, weakening or hardening the constraints on the edges of the constraint graph associated with φ. Providing an analytic form for P_φ is a difficult task, not yet solved in its generality. Nevertheless, a rough estimate can be obtained for particular cases. Thus we will consider simple formulas as building blocks of more complex formulas, for example,

$$\psi(x_1, x_2) = \bigwedge_{i=1}^{k} \alpha_i(x_1, x_2). \tag{9.10}$$

Under the assumption that all the relations α_i contain exactly N tuples independently sampled from Λ^2, the probability that a constant pair (a_1, a_2) satisfies ψ is $P_\psi = P_\alpha^k$. Thus the set of pairs of constants satisfying ψ in a universe e is the intersection of the k relational tables α_i, whose expected size (with respect to the different constraint graphs having the same parameters) is

$$N_\psi = L^2 P_\alpha^k = L^2 \left(\frac{N}{L^2}\right)^k = \frac{N^k}{L^{2(k-1)}}. \tag{9.11}$$

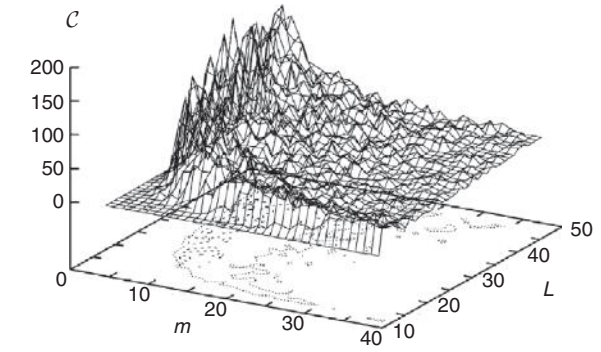
As a particular case, when $k = 2$ we have

$$N_\psi = \frac{N^2}{L^2}. \tag{9.12}$$

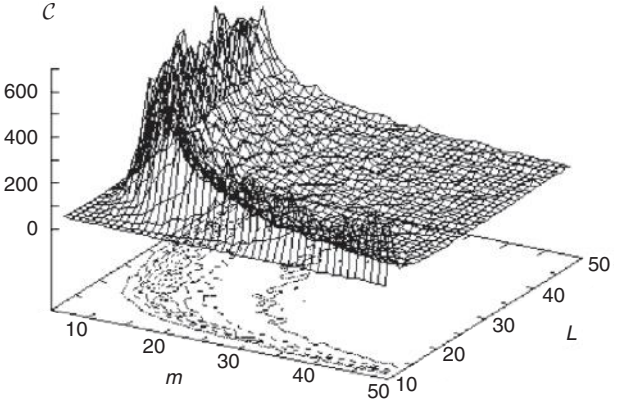
Considering model RL, we note that formulas of the type (9.10) emerge as fundamental building blocks. In fact, according to the generative algorithm described in Section 9.1.3, several literals may share the same variable pair; hence, the global constraint on an edge of the constraint graph associated with φ will be a subformula of the type (9.10).

Another construct worth considering for our purpose is a chain of two predicates through a single shared variable x_s:

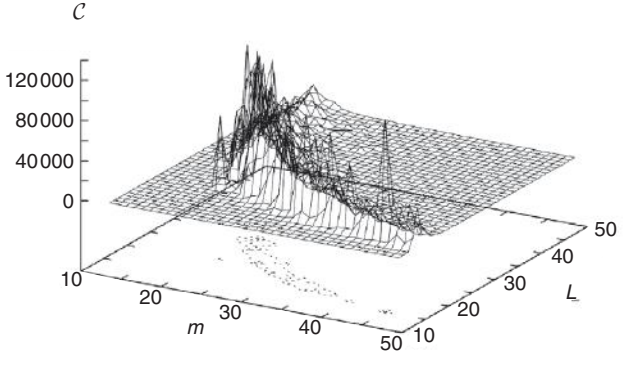
$$\varphi(x_i, x_s, x_j) = \alpha_1(x_i, x_s) \wedge \alpha_2(x_s, x_j).$$



(a)



(b)



(c)

Figure 9.10 Dependency of the complexity $\mathcal{C}$ on the number n of variables: (a) $n = 4$ and $N = 100$; (b) $n = 6$ and $N = 100$; (c) $n = 14$ and $N = 100$. Note the different scales on the $\mathcal{C}$ axis.

If all the relational tables are randomly sampled according to the same distribution from the set Λ^2, the distributions of the values of x_s occurring in R_{α_1} and R_{α_2} are each multinomial, with equal probabilities of success $1/L$ for all $a_i \in \Lambda$:

$$P_{\alpha_1}(k_1, \ldots, k_L) = \binom{N}{k_1, \ldots, k_L} \frac{1}{L^N}, \tag{9.13}$$

$$P_{\alpha_2}(h_1, \ldots, h_L) = \binom{N}{h_1, \ldots, h_L} \frac{1}{L^N}, \tag{9.14}$$

where the first factor on each right-hand side is a multinomial coefficient. The tuples that satisfy $\varphi(x_i, x_s, x_j)$ are those that occur in the natural join between the tables R_{α_1} and R_{α_2}. The average number of these tuples is given by

$$N_\varphi = \mathbb{E}\left[\sum_{i=1}^{L} k_i \, h_i\right].$$

The sum is a stochastic variable, where the k_i and h_i follow the distribution (9.13) and (9.14), respectively. Then

$$N_\varphi = \sum_{i=1}^{L} \frac{N}{L} \frac{N}{L} = \frac{N^2}{L}. \tag{9.15}$$

We are now able to explain why increasing the number n of variables allows the phase transition edge to move far from the origin of the (m, L) plane. First, considering the generic formula (9.7) generated by model RL, we notice that subformulas of type (9.10), with $k = 2$, may include either two β's or one α and one β. However, chaining certainly occurs, by construction, between the $n - 1$ α's and may additionally occur between the β's and between the α's and β's. For a given m, increasing n has the effect of reducing the average number of literals built from the same variable pair and, hence, the exponent k in (9.11) decreases. In support of this observation we report, in Figure 9.11, the probability distributions of s for $n = 20$.[6] As can be seen, the number of edges in the constraint graph is much closer to m, meaning that fewer variable-pair duplications occur.

As chaining is less restrictive than intersection, a hypothesis in which chaining dominates will be easier to verify, other things being equal, than one where intersection dominates. If the probability of intersection decreases with n then the probability of chaining increases because all the predicates β generate chaining if they are not intersections (i.e., because all the variables occur anyway in

[6] Recall that s is the number of edges in the constraint graph.

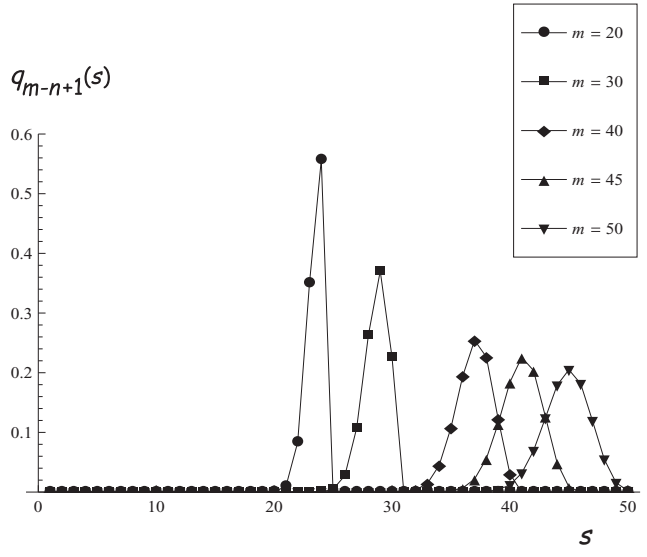


Figure 9.11 Probability distributions of the s values for $n = 20$. All distributions are shifted toward the right compared with Figure 9.4; thus variable-pair duplication is less likely. The leftmost acceptable value is $s = n - 1 = 19$.

the α's). As a consequence, the YES region is widened and the phase transition boundary moves toward the right in the (m, L) plane. Recall that, for any n, only the region to the right of the vertical line $m = n - 1$ is sampled by model RL.

9.4.2 Comparison with model B

In the literature, CSPs generated by model B are typically characterized by two parameters, namely p_1 and p_2 (see Section 4.2). For the reader's convenience, we repeat here their definitions:

$$p_1^{(B)} = \frac{2m}{n(n-1)}, \tag{9.16}$$

$$p_2^{(B)} = 1 - \frac{N}{L^2}. \tag{9.17}$$

In model RL the value of m does not correspond to the number of edges in the constraint graph, because several constraints may share the same pair of variables. Thus p_1 is a stochastic variable, because the number of edges in a CSP instance is s, whose probability distribution was computed in Section 9.1.3.

So we may expect that all quantities depending on p_1 will be, in CSPs derived from model RL, averaged over s according to the probability distribution $q_k(s)$ introduced in Section 9.1.3. In particular, the value of p_1 itself would be

$$p_1^{(RL)} = \frac{2\bar{s}}{n(n-1)}. \tag{9.18}$$

For a given value of m, p_1 assumes smaller values in model RL than in model B. In machine learning, the parameters p_1 and p_2 do not have a direct meaning. On the contrary, in machine learning 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 it contains.

In previous work, experiments have usually been done by changing p_2 and keeping p_1 constant. In (9.17) the ratio N/L^2 represents the fraction of pairs allowed to occur in the corresponding predicates. The fact that model RL allows multiple arcs on $\boldsymbol{G}$ introduces a further difference between model B and model RL: as discussed in the previous subsections, edges may be labeled by conjunctions of constraints rather than elementary constraints. Thus, even though the elementary constraints are generated with the same cardinality N, subformulas labeling the edges may have different cardinalities (smaller than or equal to the original ones). Therefore p_2 should also take into account this variation:

$$p_2^{(RL)} = 1 - \frac{\overline{N}}{L^2}, \tag{9.19}$$

where $\overline{N}$ is the average cardinality. To compute the probability distribution of the values of N in the composite tables is quite a hard task. Estimates can be obtained using arguments similar to those in Section 9.4.1. In practice, the empirical mean of the values obtained in the CSP instance under analysis can be used as an estimate.

To summarize, using the same values for m and N in models B and RL we obtain

$$p_1^{(RL)} \leqslant p_1^{(B)} \qquad \text{and} \qquad p_2^{(RL)} \geqslant p_1^{(B)}. \tag{9.20}$$

The number of "hypotheses" (constraint graphs) generated by model B is

$$|\mathcal{H}_B| = \binom{\frac{n(n-1)}{2}}{m},$$

and the number of "examples" (sets of tables) is

$$|\mathcal{E}_B| = \binom{L^2}{N}^m.$$

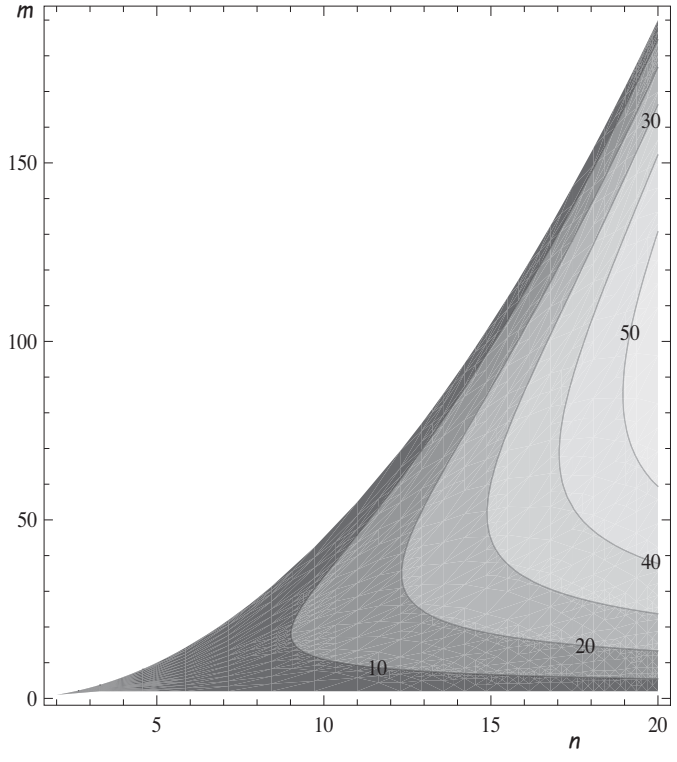


Figure 9.12 Contour plot of the logarithm to the base 10 of the number of hypotheses $|\mathcal{H}_B|$ generated by model B as a function of n and m. The value of m is upper-bounded by $n(n-1)/2$.

Comparing $|\mathcal{H}_B|$ and $|\mathcal{E}_B|$ with the corresponding values, $|\mathcal{H}_{RL}|$ in (9.8) and $|\mathcal{E}_{RL}|$ in (9.9), for model RL we note that the number of examples is the same. In Figures 9.12 and 9.13 the contour graphs of the base-10 logarithms of $|\mathcal{H}_B|$ and $|\mathcal{H}_{RL}|$ are reported, respectively. As is apparent from the figures, the spaces of hypotheses generated by the two models are rather different, not only in cardinality but also in the sampled regions of (n, m) space.

Starting from a theoretical analysis presented by Williams and Hogg (1994), Prosser (1996) derived an estimate for the critical value of p_2:

$$\hat{p}_{2,cr}^{(B)} = 1 - L^{-2/[p_1(n-1)]} = 1 - L^{-n/m}. \tag{9.21}$$

From (9.21) the following value of $\hat{m}_{cr}$ may be derived:

$$\hat{m}_{cr}^{(B)} = \frac{n \ln L}{\ln(L^2/N)}. \tag{9.22}$$

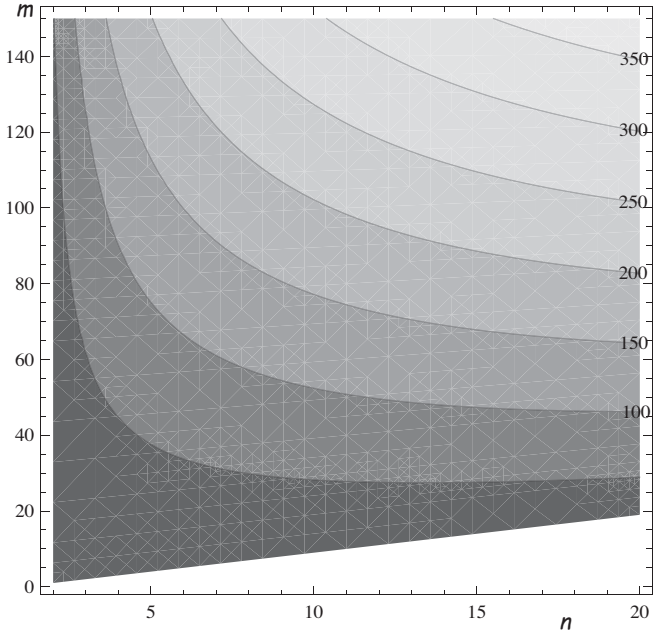


Figure 9.13 Contour plot of the logarithm to the base 10 of the number of hypotheses $|\mathcal{H}_{RL}|$ generated by model RL as a function of n and m. The value of m is lower-bounded by $n - 1$.

The same estimate as (9.21) was obtained by Smith and Dyer (1996) using a slightly different approach. Moreover, the same expression is also obtained by using the parameter κ (see (4.10)) introduced by Gent and Walsh (1996), and setting it to 1.

For model RL, $\hat{m}_{cr}$ must be replaced by the critical value of the average of s, namely $\hat{\bar{s}}_{cr}$:

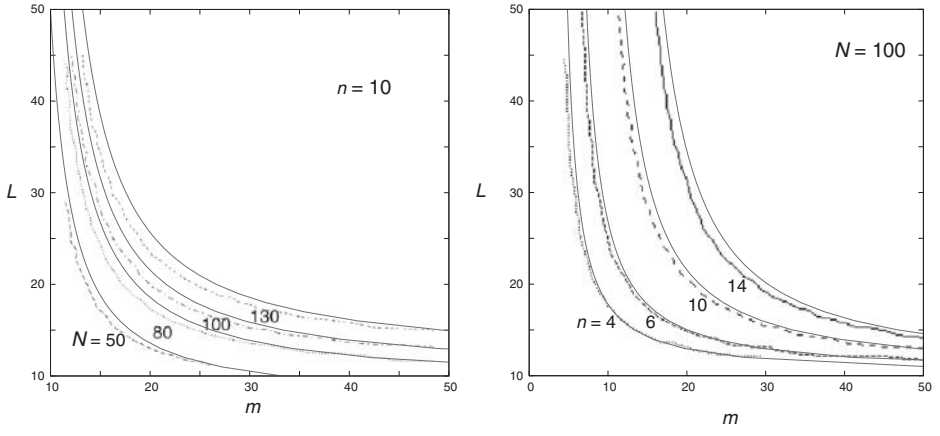
$$\hat{\bar{s}}_{cr} = \frac{n \ln L}{\ln(L^2/\overline{N})}. \tag{9.23}$$

Using (9.21) we can also compute the corresponding estimate for the critical value of $\hat{L}_{cr}$, keeping all other parameters constant. We have

$$1 - \frac{\overline{N}}{\hat{L}_{cr}^2} = 1 - \hat{L}^{-n/\hat{\bar{s}}_{cr}}.$$

The predicted critical value of L_{cr} is thus

$$\hat{L}_{cr} = \overline{N}^x, \tag{9.24}$$



(a) (b)

Figure 9.14 The values predicted for $\hat{s}_{cr}$ by (9.23) and the experimental findings shown in Figure 9.8: (a) $N = 50$, 80, 100, and 130 and $n = 10$; (b) $n = 4$, 6, 10, and 14 and $N = 100$. The upper line in each pair corresponds to the theoretical prediction of the phase transition edge. The lower line in each pair corresponds to the experimental findings already reported in Figure 9.8.

Theoretical predictions obtained from model B closely resemble the experimental findings obtained with model RL.

where $x = (2 - n/\hat{\bar{s}}_{cr})^{-1}$. It is of interest to compare the plot of function (9.22), for different values of N and n, with the experimental results shown in Figure 9.8. This is done in Figure 9.14, where substantial agreement with the critical value predicted by model B, as computed by (9.23), can be seen. However, there are small systematic differences whose explanation is intriguing. We notice that the YES region found in the empirical analysis is a little smaller than that predicted by model B. However, the curves overlap when L is small, in the region close to the m axis. This also occurs for larger m values.

To find an interpretation of this phenomenon, two aspects have to be considered. First, owing to the difference between m and $\bar{s}$, each experimental curve must be to the left of the theoretical one because it corresponds to the actual value of $\bar{s}$; this value is not known in advance (we only know m), but certainly $\bar{s} \leqslant m$. Moreover, we need to consider how the structure of a formula may affect the probability of success of the covering test. Model B does not take into account the structure of the constraint graph, because the tightness of the constraints is the same for all edges. In model RL the latter is not true because many literals can share the same variable pair. In order to see the

Table 9.1 Expected number of tuples for subformulas in φ_1 and φ_2

Formula	Relation	Expected number
φ_1	$\alpha_1(x_1,x_2) \wedge \beta_2(x_1,x_2)$	$N_{\alpha_1,\beta_2} = N^2/L^2$
φ_1	$\alpha_3(x_2,x_3) \wedge \beta_4(x_2,x_3)$	$N_{\alpha_3,\beta_4} = N^2/L^2$
φ_2	$\alpha_1(x_1,x_2) \wedge \beta_2(x_1,x_2) \wedge \beta_3(x_1,x_2)$	$N_{\alpha_1,\beta_2,\beta_3} = N^3/L^4$
φ_2	$\alpha_4(x_2,x_3)$	N

effect of this difference on the phase-transition edge, let us first consider an example.

EXAMPLE

Let φ_1 and φ_2 be the formulas

$$\varphi_1 = \alpha_1(x_1,x_2) \wedge \beta_2(x_1,x_2) \wedge \alpha_3(x_2,x_3) \wedge \beta_4(x_2,x_3) \tag{9.25}$$

and

$$\varphi_2 = \alpha_1(x_1,x_2) \wedge \beta_2(x_1,x_2) \wedge \beta_3(x_1,x_2) \wedge \alpha_4(x_2,x_3). \tag{9.26}$$

Both formulas have the same number of literals. However, the literals in φ_1 are equally distributed over the variable pairs (x_1,x_2) and (x_2,x_3) while in φ_2 three literals share (x_1,x_2) and only one is applied to (x_2,x_3).

Let us estimate the expected number of solutions in the domain Λ^3 for the formulas φ_1 and φ_2 in the above example. The expected numbers of tuples satisfying the subformulas of φ_1 and φ_2 are given in Table 9.1. Using expressions (9.11) and (9.15), we obtain for N_{φ_1} and N_{φ_2}, the exptected number of tuples respectively satisfying φ_1 and φ_2,

$$N_{\varphi_1} = \frac{1}{L}\,\frac{N^2}{L^2}\,\frac{N^2}{L^2} = \frac{N^4}{L^5}, \tag{9.27}$$

$$N_{\varphi_2} = \frac{1}{L}\,\frac{N^3}{L^4}\,N = \frac{N^4}{L^5}. \tag{9.28}$$

Constraints in model RL are tighter than in model B for large values of L.

Thus the expected numbers of solutions for φ_1 and for φ_2 are the same. This means that, on averaging the number of solutions obtained from a large number of constraint graphs (all having the same parameters), we will obtain approximatively the same value. However, we do not know what happens for every single graph; the number of solutions could always be close to the average or there

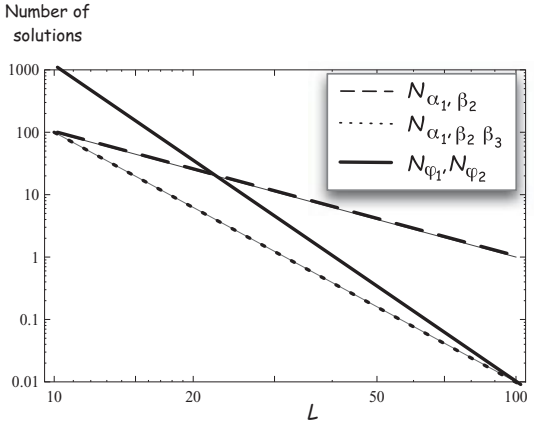


Figure 9.15 Average number of solutions for φ_1 and φ_2 and two of their subformulas, for $N = 100$.

could be no solution for many graphs and a large number of solutions for a few graphs. For this reason it is important to consider the variance of the number of solutions in addition to the average. In fact, this difference in behavior actually occurs for φ_1 and φ_2.

Figure 9.15 shows plots of the expected numbers of solutions N_{φ_1}, N_{φ_2}, N_{α_1,β_2}, $N_{\alpha_1,\beta_2,\beta_3}$ (see Table 9.1) as functions of L. For $L = 50$, $N_{\varphi_1} = N_{\varphi_2} = 10$. However, in the case of φ_1 this number was obtained from the natural join of two tables each containing the expected number of 10 tuples while in the case of φ_2 the number was obtained by the natural join of a table that contains an average number of 0.1 tuples and a table with 100 tuples. This means that in the latter case we can expect that in 90% of cases we will have no solution while in 10% of cases we will have almost 100 solutions. Thus φ_2 is on the border of the NO region while φ_1 is on the border of the YES region. In other words, formulas where literals are irregularly distributed over the variables (i.e., the number of literals defined on the same variable pair is highly unpredictable across pairs) are globally more difficult to satisfy.

As model RL may generate formulas with literals irregularly distributed on the variables, we can expect a number of cases where the probability of success is smaller than for model B (given the same m), even if the expected number of solutions is the same.

In support of this explanation we observe that when n is small the differences between the theoretical prediction of model B and the experimental findings of our model are less conspicuous. In fact, when the variables are few the literals

have less chance of being distributed irregularly on them. Moreover, when L is small the constraints due to the predicates are more soft, and so the effect of the difference in structure in the formulas becomes less evident.

9.4.3 Asymptotic behavior and model RB

As discussed in Section 4.3.1, Achlioptas *et al.* (1997) demonstrated that, when the number n of variables grows to infinity, the probability that a random instance of CSP generated by model B is unsatisfiable tends to 1, provided that $p_2 > 1/L$ and that the number of constraints is $m = \mathcal{O}(n)$.

In model RL there is a substantial difference from model B, because in model B we must have $m \leqslant n(n-1)/2$ whereas in model RL we must have $m \geqslant n-1$, and, for the number of edges in the constraint graph, $s \leqslant n(n-1)/2$. Then, given constant $m, N,$ and L, model B can generate a CSP for $n \to \infty$, whereas model RL cannot. It is thus necessary that $m \to \infty$ when $n \to \infty$ merely for the generation procedure of model RL to be applicable. In order to maintain the basic feature of model RL, namely that the generated formula is connected, we must have $m = \Omega(n)$; if m grows linearly with n, i.e., $m = \Theta(n)$, since the connected part always includes $n-1$ literals the graph consists substantially of this connected part plus some other edges.[7] As the expected number of satisfying tuples for a chain of k predicates is given by formula (9.15), the probability of solution for a chain of $n-1$ literals is in turn given, on average, by the number of satisfying tuples divided by the number of possible tuples, namely

When both n and m grow to infinity, formulas become always unsatisfiable.

$$P_{sol} = \left(\frac{\overline{N}}{L^2}\right)^{n-1}. \tag{9.29}$$

When $n \to \infty$, probability (9.29) tends to 0. Clearly adding more constraints makes the problem even harder. In order to obtain regions of asymptotic satisfiability, the number of constants in the domain must increase with n, as in model RB (see Section 4.3.2). In this last model it is of interest to note that the critical value r_{cr}, given by Theorem 4.2, formally coincides with expression (9.22) even though the parameters have different definitions:

$$r_{cr} = -\frac{\alpha}{\ln(1-p)} = \frac{\ln L}{\ln n \ln(L^2/N)} = \frac{\hat{m}_{cr}}{n \ln n}. \tag{9.30}$$

Formula (9.22) can be immediately derived from (9.30).

[7] For Ω and Θ, see the discussion after equation (1.4).

9.5 Smart algorithms for the covering test

In previous sections we have observed the emergence of phase transitions in the covering test even for a small number of variables. Consequently, we have measured the complexity for searching in the phase transition region using a backtrack stochastic search algorithm. This approach gave us the baseline for the problem complexity. However, in Chapter 4, we described how modern CSP solvers exploit efficient heuristics that allow them to solve very large problems in a relatively short time. We may then wonder whether the complexity shown in Figure 9.9 could possibly be altered using one of these advanced algorithms.

As an answer we will provide results obtained using two algorithms based on CSP methods similar to those described in Chapter 4. One is multilevel coordinate search (MCS), from Scheffer *et al.* (1996) and the other is Django, from Maloberti and Sebag (2004).

The MCS algorithm exploits a constraint graph to prune the search space using the techniques of constraint propagation. The constraint graph contains both clauses with variables and also ground literals. Thus θ-subsumption (the covering test) is transformed into the problem of finding a maximal *clique* (a complete graph) in the constraint graph. As this problem is of exponential complexity, the MCS algorithm searches for cliques of size n for pruning the search space. Afterwards, the maximal clique is searched for by starting from the cliques of size n that have been found already.

Using smart heuristics from the CSP domain, order-of-magnitude decreases in the complexity peak are observed.

The Django algorithm also exploits graph constraint propagation, such as arc consistency, in order to prune the search space. The major difference from MCS and other matching algorithms is that Django works on a constraint graph obtained by transforming the original CSP into a *dual* binary CSP.

In the following we will briefly review the technique used for this transformation. The binary CSP is defined at a metalevel with respect to the original problem. Let φ denote the formula to be tested and U the universe where the test is to be performed. For every literal $p_i(x_i, \ldots, x_j)$ occurring in φ, a metavariable Y_i is defined whose domain is the relation R_i defining the semantics of p_i in U, i.e., tuples in R_i are the values that Y_i can assume. Metavariables corresponding to literals built on the same predicate symbol p_i share the domain R_i. Metavariables are constrained by metarelations. For each literal pair $p_i(\ldots, x_r, \ldots), p_k(\ldots, x_r, \ldots)$ sharing at least one variable x_r, a binary constraint $r_j(Y_i, Y_k)$ is defined: this states that values for Y_i and Y_k must correspond to legal substitutions for the subformula $p_i(\ldots, x_r, \ldots) \wedge p_k(\ldots, x_r, \ldots)$, i.e., variables shared between the two literals must be bound to the same constant. An example of the metavariables, metaconstraints and constraint graph obtained

Django maps the original CSP into a binary CSP.

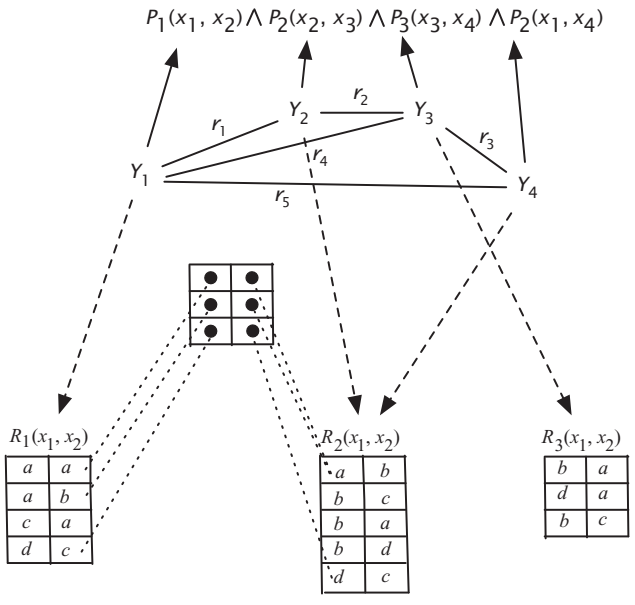


Figure 9.16 Example of the binary CSP constructed by the algorithm Django before executing the covering test of a formula.

from a formula using four literals is provided in Figure 9.16. In this example, for the sake of simplicity the original CSP was also binary; however, the same method can be applied to formulas containing literals of any arity. On the constraint graph obtained in this way it is simple to test for arc consistency and for deterministic matching, i.e., literals for which there exists only one admissible match.

The performances of MCS and Django are illustrated for comparison in Figures 9.17 and 9.18, which show the match complexity for the same region of the space (n, m, N, L) as in Figure 9.9(a). It is surprising how MCS already exhibits an advantage of several orders of magnitude in the phase-transition region over the basic backtracking algorithm (it is about 200 times faster). In turn, Django has an advantage of about one order of magnitude over MCS (it is 2000 times faster than stochastic backtracking). Interestingly, both MCS and Django yield the same pattern for the probability of solution and for the complexity as that shown in Figure 9.7. The only difference is that for these algorithms the height of the complexity peak is much lower. One may see, therefore, that the phase transition exists independently of the search algorithm, as expected.

The phase transition exists independently of the search algorithm.

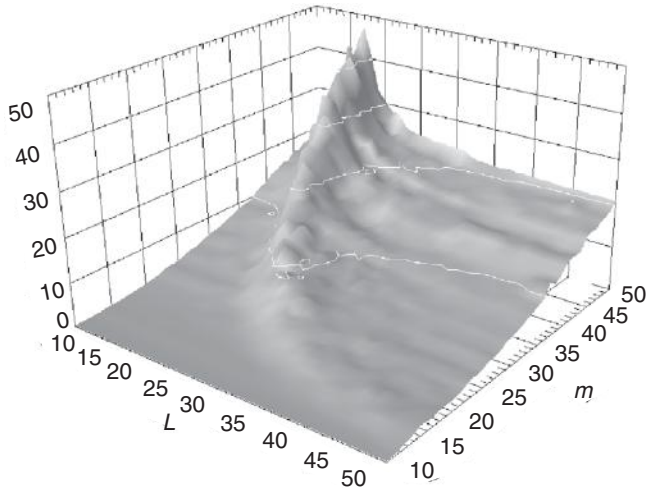


Figure 9.17 Three-dimensional plot of the complexity reported by MCS as a function of the time elapsed in seconds (vertical axis), for $n = 10$ and $N = 100$, vs. m and L.

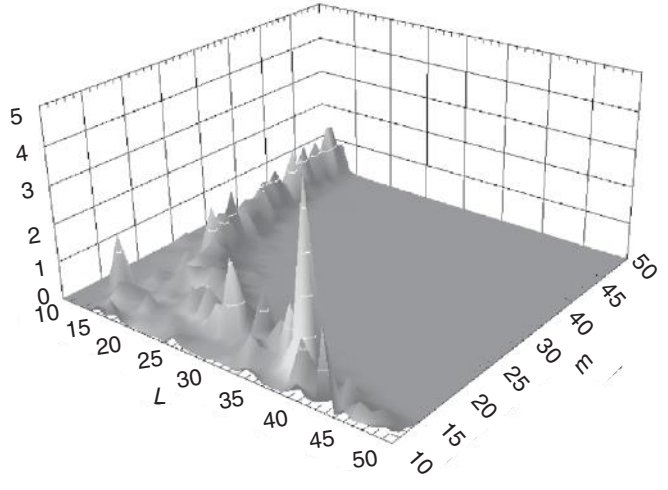


Figure 9.18 Three-dimensional plot of the complexity reported by the algorithm Django as a function of the time elapsed in seconds (vertical axis), for $n = 10$ and $N = 100$, vs. m and L.

9.6 Comments

As relational learning relies heavily on matching candidate hypotheses to training examples, investigation of the matching problem's characteristics is of fundamental relevance for the success of this type of learning. Owing to the straightforward equivalence between a matching problem and a CSP, one may expect that the former shows the presence of a phase transition in the same way as the latter. What is relevant, in the context of learning, is the identification of suitable control parameters and the location of the phase transition within the parameter space. As we have seen, a simple adoption of the classical parameters p_1 and p_2 is not well suited to learning because they cannot be associated with recognizable features, either of the hypothesis (the formula) or the examples. The parameters selected (the number of variables n and the number of predicates m for the formula, the number of goods N, and the number of constants L for the examples), beyond being meaningful in learning allow comparisons to be made with other models, as they can be expressed as functions of p_1 and p_2.

As the investigation of all four parameters at the same time is computationally prohibitive, we have chosen to focus on two, namely m for the formula and L for the examples, keeping n and N fixed. However, a limited exploration has highlighted the fact that analogous results are obtained by choosing any of the pairs $(n, L), (n, N), (m, L)$, and (m, N) as control parameters. Concerning the location of the phase transition, it turns out that the set of critical pairs (m_{cr}, L_{cr}) is situated in a region of the parameter space well populated by the learning problems encountered in practice. This aspect will be discussed in more detail in the next chapter.

Regarding problem generation, model B, used for CSPs, and model RL, used for matching problems, produce different ensembles of instances, i.e., the hypothesis spaces $\mathcal{H}_B$ and $\mathcal{H}_{RL}$ are not the same whereas the two models generate the same sets of relational tables. The reason why we have not used model B directly is that formulas in $\mathcal{H}_B$ are not realistic in learning problems, where learned formulas with several predicates on the same variable are quite common. Moreover, the number m of predicates occurring in a formula of model RL is not upper-bounded by $M = n(n-1)/2$, so that long concepts may also be considered without the need to increase the number n of variables. As a consequence, there is a region where formulas can be generated only by model B and another where formulas can be generated only by model RL. These regions can be clearly seen in Figures 9.12 and 9.13. Even for a pair (n, m) located in the region where both models can generate instances, the sets of formulas generated by the two models are different. Both use m predicates, but while model B cannot produce formulas with predicates defined on the same pair of variables (whereas model RL can), model RL cannot produce formulas in which not all

variables pairs $(x_1, x_2), \ldots, (x_{n-1}, x_n)$ occur (whereas model B can). Using the probability distribution $q_k(s)$, we can see that the probability that model RL will generate a formula with all pairs of variables different is

$$q_{m-n+1}(m) = \prod_{i=1}^{m-n+1} \left(1 - \frac{n+i-2}{M}\right) \qquad (n \leqslant m \leqslant M),$$

which, for a given n, decreases with increasing m.

We may notice that model RL generates constraint graphs that are the superposition of a path between node x_1 and node x_n including n nodes and $n-1$ edges and a random graph with the same nodes and $s-n+1$ edges. However, this random graph does not belong to $\mathcal{G}_{n,s-n+1}$ because the edges are extracted with replacement in $m-n+1$ trials.

A comparison of model RL with model RB is more tricky, because one has to choose what is kept constant and what may vary. Let us suppose that n and N are constant, as in model RL, and let $k = 2$. Then let (m_{cr}, L_{cr}) be a point on the phase transition for model RL. The corresponding value of r will be

$$r = \frac{m_{cr}}{n \ln n},$$

and the corresponding value of α will be $\ln L_{cr}/\ln n$. Finally, we obtain $p = 1 - N/L_{cr}^2$. We have to compare the obtained r value with the critical value

$$r_{cr} = -\frac{\alpha}{\ln(1-p)}.$$

In order for r_{cr} to be the location of the phase transition, we must have $\alpha > 1/2$ and $p \leqslant 1/2$, namely:

$$\sqrt{n} < L_{cr} \leqslant \sqrt{2N}.$$

If the above condition holds then r_{cr} is indeed the location of the phase transition and the computed value of r has to be compared with it, in order to check the probability of solution.

EXAMPLE

In order to see how the problems generated by models RB and RL, respectively, are mapped, let us take $n = 10$ and $N = 100$. One point on the phase transition for model RL is given by $m_{cr} = 30$, $L_{cr} = 14$. From these values we obtain

$$r = 1.303, \qquad \alpha = 1.146, \qquad p = 0.490.$$

The conditions to be verified are $\alpha > 1/2$, which is true, and $p \leqslant 1/2$, which is also true. The critical value for r is $r_{cr} = 1.45$. The actual value

of r is lower than this. Thus in this case a problem generated by model RB, with the parameters of a problem generated by model RL at the phase transition, is located in the solvable part of the plane even though it is close to the phase transition. This result is not generalizable, and its only purpose is to show a way of obtaining the mapping. By changing the point selected on the phase-transition line of model RL or by varying n and N, different situations may arise.

Finally, considering the computational cost, we note that, even though the absolute cost varies according to the search algorithm used, the same behavior emerges: all tested algorithms show the pattern *easy–hard–easy* across the phase transition region.

10

Phase transitions and relational learning

Contents

In the previous chapter we showed how the covering test in relational learning exhibits a phase transition associated with a complexity peak, for control parameter values typical of the problems investigated by current relational learners. We also showed that the complexity associated with the phase transition in matching can be partially tamed using smart search algorithms. However, as soon as the number of variables increases a little (say, from four to five) the complexity is again a strongly limiting factor for learning, because a learner must face hundreds of thousands of matching problems during its search for hypotheses (formulas).

Leaving aside the problems caused by the computational complexity of matching, one may wonder whether the presence of a phase transition has additional effects on learning, for instance whether it affects the quality of the learned knowledge. Another question is whether it is possible to escape from

the region of the phase transition by suitably manipulating the control parameters. In this chapter we try to provide an answer to these questions, by means of an experimental analysis and its interpretation.

10.1 The experimental setting

In order to test the independence of the results from the learning algorithm, we used the learners FOIL (Quinlan and Cameron-Jones, 1993), SMART+ (Botta and Giordana, 1993), G-Net (Anglano *et al.*, 1997; Anglano and Botta, 2002), and PROGOL (Muggleton, 1995) described in Chapter 6.

To be more precise, the impact on the quality of learning of the phase transition in the covering test will be analyzed under three aspects:

The impact of the phase transition on relational learning is analyzed with respect to the difficulty in solving the task, the effectiveness of the heuristic used to guide the inductive search, and the accuracy of the learned concept descriptions.

- Can "easy" and "difficult" learning regions be distinguished, with respect to the control parameters of the matching problem? How are these regions located in relation to the phase transition? Do other phase transitions exist that are not related to the covering test?

- Where and when are common search criteria, involving for example the information gain principle (Quinlan, 1990) or the minimum description length principle (Muggleton, 1995), reliable? What is the impact of plateaus?

- For a learning problem, the generalization error is certainly an important aspect but it is not the only one. Another is the meaningfulness of the acquired concept description; this is particularly relevant for automated knowledge discovery, where the learned knowledge should provide the domain experts with new, relevant, insights. Thus another question to be answered is to what extent the acquired concept coincides with the target concept, beyond its prediction ability.

The above questions have been answered experimentally. The artificial problem generator described in Section 9.1.3 has been extended to generate fully relational learning problems *with known target concept*. A suite of about 500 problems was constructed, sampling the YES, NO, and phase transition regions. Results were obtained using the three relational learners FOIL 6.4, SMART+, and G-Net. PROGOL was not able to cope with the computational complexity of the task. As we will discuss in the following, these systematic experiments shed some light on the behavior, potentialities, and limitations of existing relational learners.

10.1.1 Generating artificial learning problems

We will now review the procedure used to construct a set of artificial learning problems (Giordana *et al.*, 2000a; Botta *et al.*, 2003). A relational learning problem Π is a triple $(c, \mathcal{S}_L, \mathcal{S}_T)$, where c is the target concept and $\mathcal{S}_L$ and $\mathcal{S}_T$ are the learning and test sets, respectively.

For the sake of simplicity, concepts were restricted to be binary, described by conjunctive formulas, as generated by model RL (see Section 9.1). This means that a target concept (relation) description has the form

$$p_1(x_{i_1}, x_{j_1}) \wedge p_2(x_{i_2}, x_{j_2}) \wedge \cdots \wedge p_m(x_{i_m}, x_{j_m}) \rightarrow c, \tag{10.1}$$

where the target concept c is a constant name and the variables $(x_{i_k}, x_{j_k})(1 \leqslant k \leqslant m)$ are taken from a set $\mathbf{X} = \{x_j \mid 1 \leqslant j \leqslant n\}$ and are implicitly existentially quantified. In the target concept there are no repeated predicate names and no variable pairs made up by two instances of the same variable, i.e., $x_{i_k} \neq x_{j_k}$.

Learning problems are generated in three steps.

The procedure for generating a learning problem consists of three steps. In the first step a concept description is generated, using the algorithm described in Section 9.1 for sampling the hypothesis space generating covering test problems. Then each concept description c is stochastically built up, by model RL, using the variable set $\mathbf{X}$ and m literals. All literals are built on distinct predicate symbols. Complementary experiments (not reported here) show that this restriction does not affect the results to any significant extent. In order to keep the computational cost within reasonable limits, the number n of variables is set to 4 ($n = 4$) in all target concepts.[1]

If $n = 4$, from (9.8) we obtain that the number of formulas that Model RL can generate is:

$$|\mathcal{H}_{RL}| = \binom{m}{3} \frac{12^m}{288} = \mathcal{O}(m^3 12^M).$$

Learning set and test set

Next the learning set $\mathcal{S}_L$ and the test set $\mathcal{S}_T$ are created, using a variant of the procedure we described for generating the examples for the covering test. Thus each example e is defined by a set $\mathbf{R}$ of relational tables, sampled without replacement from a set $\Lambda \times \Lambda$ of pairs of constants. Each relation R_i defines the semantic (extension) of a corresponding predicate p_i occurring in the concept description. An example e is labeled as positive if it verifies the definition of c, and as negative otherwise. Again for computational reasons, the size N of each

[1] Note that most relational learners have similar restrictions. For instance, in the mutagenesis domain (King *et al.*, 1995), the maximum number of chemical atoms considered in a hypothesis, corresponding here to the number of distinct variables, varied from 3 to 5.

R_i is set to 100 for all relations, and the size L of Λ is taken to be the same for all examples in the same problem Π. As the number of examples generated by model RL does not depend on n, this last is the same as in (9.9).

A clarification is needed with respect to the generation of the examples. In each example we have to choose the number of tables to be associated with it. When we were studying the matching (φ, e) between a hypothesis and an example, the natural number of tables to be associated with e was the number of predicates in φ. In fact, additional predicates would be ignored by the matching procedure, and some missing predicate tables would make the hypothesis automatically false (according to the "closed world assumption", no tuple verifies the predicate).

When generating a learning problem the target concept c is known, and hence the number m of its predicates as well. However, during learning the examples are likely to be matched against hypotheses of different lengths; then the determination of the number of tables in the example is not obvious. We have chosen to describe each example by m tables, corresponding to the number of predicates in c. The reason is that in relational learning the predicates available to construct hypotheses are given, and they constitute the "background knowledge". These predicates have for relational learning the same role that attributes have for propositional learning. Thus, the learner only constructs hypotheses with literals built upon the given predicates.[2]

Furthermore, it is worth noticing that sampling the set $\Lambda \times \Lambda$ does not guarantee that every relational table contains all the constants in Λ. As a consequence, the L value for each relation R_h, which is the number of constants appearing in R_h, may be different from relation to relation; the only observed effect is a slight increase in the apparent width of the phase transition, as discussed later in the chapter.

A set of 451 artificial relational learning problems was generated to analyze the impact of the phase transition on relational learning.

In order to visit the YES, NO, and phase transition regions as uniformly as possible, while avoiding an exhaustive exploration, 451 (m, L) pairs were uniformly selected without replacement, where m ranged in the [5, 30] interval and L ranged in the [12, 40] interval.

For each selected pair (m, L) a learning problem $\Pi_{m,L}$ was constructed, as explained above. As anticipated, the procedure for constructing the training and test examples is a variant of that described in Section 9.1. The modification was made necessary by the following difficulty: if (m, L) lies in the YES region (on the left of the phase transition) then by construction c will almost surely cover any stochastically constructed example. In other words, the training and test sets would contain a huge majority of positive examples (the ratio of positive and

[2] We do not consider here the case of *constructive learning*, in which new predicates are introduced during learning.

Procedure ProblemGeneration (m, L)

Construct c with m literals using model RL
$\mathcal{S}_L$ = DataGeneration(m, L, c).
$\mathcal{S}_T$ = DataGeneration(m, L, c).
Return $\Pi = (c, \mathcal{S}_L, \mathcal{S}_T)$.

Procedure DataGeneration (m, L, c)

$np = 0$, $ne = 0$
Let $\mathcal{S} = \emptyset$
while $ne < 100$ **or** $np < 100$ **do**
  Generate a random example e with model RL
  if e is covered by c
    then **if** $np = 100$
      then ChangeToNegative(c, e)
        Set label of e = NEG
      else Set label of e = POS
    endif
    else if $ne = 100$
      then ChangeToPositive(c, e)
        Set label of e = POS
      else Set label of e = NEG
    endif
  endif
  $\mathcal{S} = \mathcal{S} \cup \{(e, \text{label})\}$
  if label = POS **then** $np = np + 1$
    else $ne = ne + 1$
endwhile
Return $\mathcal{S}$

Figure 10.1 Generation of the training and test sets of examples.

negative examples could be as high as 10^6). Symmetrically, if (m, L) lies in the NO region (on the right of the phase transition), the training and test sets would contain a huge majority of negative examples. The generation of the examples thus needs to be made according to the procedure given in Figure 10.1.

Randomly generated examples are modified in order to obtain balanced learning and testing sets.

The procedure **ProblemGeneration** first constructs the target concept c; then the training and test sets are built by the procedure **DataGeneration**. The latter

Procedure ChangeToPositive(c**,** e**)**

Uniformly select four constants in the domain Λ of the variables, and denote them (without loss of generality) a_1, a_2, a_3, a_4
Let θ denote the substitution $\theta = \{x_1/a_1,\ x_2/a_2,\ x_3/a_3,\ x_4/a_4\}$
for each literal $p_k(x_i, x_j)$ in c, **do**
    if pair (a_i, a_j) does not already belong to table R_k
        then replace a uniformly selected pair in R_k with (a_i, a_j)
    endif
end
Return e

Figure 10.2 Procedure to turn a negative example of the concept c, generated stochastically by model RL, into a positive example. It is sufficient to select a consistent substitution for the four variables and add the corresponding pairs to the appropriate tables, at the same time removing a randomly selected pair.

accumulates examples constructed with model RL; the examples are labeled positive or negative, respectively, depending on whether they verify c. When the maximum number of positive (respectively, negative) examples is reached, further examples are prepared using the **ChangeToNegative** (respectively, **ChangeToPositive**) procedure, to ensure that the training and test sets are equally distributed. The procedure **ChangeToPositive** turns a negative example e into a positive one and is given in Figure 10.2. Conversely, the procedure **ChangeToNegative** turns a positive example e into a negative one and is given in Figure 10.3.

10.1.2 The learners

The learners for the experiments were selected from the relational learners that we briefly reviewed in Chapter 6. Three learning strategies were considered: a top-down depth-first search, a top-down beam search, and a genetic algorithm-based search.

The experimentation was performed using three learners, FOIL, SMART+, and G-Net.

The majority of the learning experiments were done using the top-down learner FOIL (Quinlan and Cameron-Jones, 1993). FOIL starts with the most general hypothesis and iteratively specializes the current hypothesis h_t by adding as a conjunct the "best" literal $p_k(x_i, x_j)$ according to some statistical criterion, involving for example the information gain (Quinlan, 1986, 1990) or the minimum description length (MDL) (Rissanen, 1978). When further specializations

Procedure ChangeToNegative(c**,** e**)**

Build up the set Θ of all substitutions
$\theta = \{x_1/a_1,\ x_2/a_2,\ x_3/a_3,\ x_4/a_4\}$ such that e verifies c
Randomly select a literal $p_k(x_i,\ x_j)$ in c
Remove from relation R_k in e all pairs $(\theta(x_i), \theta(x_j))$ of constants belonging to a substitution θ in Θ
Replace them by uniformly selected pairs of constants not belonging to any substitution θ in Θ
Return e

Figure 10.3 Procedure to turn a positive example of the concept c, generated stochastically by model RL, into a negative one. It is necessary to disrupt the concatenation of pairs that satisfies the concept. Taking any literal, all pairs that belong to a substitution in Θ are removed and replaced by pairs that do not belong to any substitution in Θ. This process may occasionally introduce a new matching tuple; then the new pairs must be checked out. In principle, the transformation of a positive into a negative example may turn out to be impossible; in this case the example has to be discarded. However, in the experiments that were performed this case never occurred.

of the current hypothesis do not lead to further improvements h_t is selected, all positive examples covered by h_t are removed from the training set, and the search is restarted unless the training set is empty. The final hypothesis $\hat{c}$ returned by FOIL is the disjunction of all selected, conjunctive, partial hypotheses h_t.

We must note that the space of hypotheses explored by FOIL does not exactly coincide with that generated by model RL. Both FOIL and model RL use only the m predicates provided as background knowledge, but FOIL allows formulas with multiple occurrences of the same predicate having different arguments. Moreover, FOIL allows literals having the same variable as both first and second argument. However, FOIL's formulas are connected, as are those generated by model RL, because a new literal $p_k(x_i, x_j)$ must share at least one variable with the part of the formula already built up. Thus, for a given m, the set $\mathcal{H}_{RL}$ of formulas generated by model RL is a subset of those of the same length generated by FOIL. More precisely, model RL can generate formulas with a length t such that $n - 1 \leqslant t \leqslant m$ (in the specific case considered, $3 \leqslant t \leqslant m$), whereas FOIL builds up, in principle, formulas with a length t such that $1 \leqslant t \leqslant mn^2$ (in the specific case considered, $1 \leqslant t \leqslant 16m$). Therefore, even for $n = 4$, FOIL allows hypotheses with a lesser number n' of variables ($1 \leqslant n' \leqslant n$).

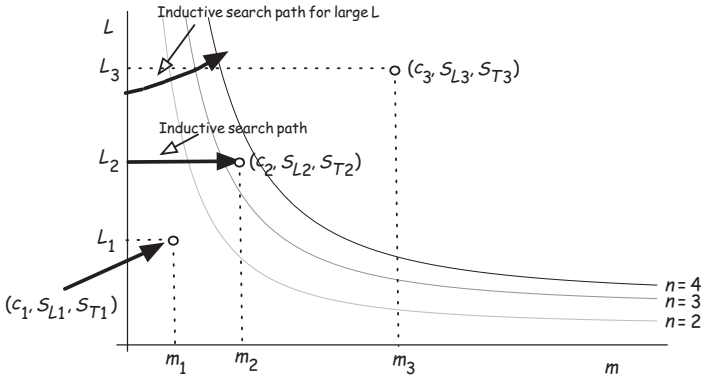


Figure 10.4 Location of learning problems in the (m, L) plane. Top-down learners visit candidate hypotheses from left to right.

In order to make the results of Chapter 9 applicable to the discussion of learning, we need to consider, in the (m, L) plane, multiple phase-transition curves, i.e., those corresponding to $n' = 2, 3, 4$, and use the curve associated with the number of variables in FOIL's current hypothesis (see Figure 10.4). An exception occurs when $m = 1$. In this case model RL cannot generate any hypothesis, because at least two variables are needed to construct the first literal $\alpha(x_1, x_2)$, as described in Section 9.1.3; in fact, $m = 1$ would imply $n = 2$, as the same variable cannot occur, in formulas generated via model RL, in both arguments of a literal. Nevertheless, when FOIL generates its first hypothesis with one literal, $h_1(x_1, x_2) = p_k(x_1, x_2)$, this hypothesis is certainly verified in all examples, both positive and negative, as long as the corresponding table in every example is not empty.

SMART+

Another top-down learner, SMART+ (Botta and Giordana, 1993), has also been used. The main difference between FOIL and SMART+ resides in their search strategies; FOIL basically performs hill climbing and uses a limited amount of backtrack, whereas SMART+ uses a beam search with a user-supplied beam width. The search space visited by FOIL or SMART+ can be visualized as a path in the (m, L) plane (see Figure 10.4). Both learners navigate in the plane by moving from left to right, as the number t of literals in the current hypothesis is incremented at each step.

FOIL and SMART+ are based on a top-down learning strategy. G-Net is based on a genetic algorithm.

When there are only a small number of constants in a variable's domain Λ, the relational tables for the examples in the learning set $\mathcal{S}_L$ are likely to involve all possible constants. Top-down learners therefore navigate the horizontal line $L = |\Lambda|$ in the (m, L) plane. For a large number of constants, it might happen that not all constants appear in all relational tables. Thus the number of *effective*

constants considered in early learning stages might be less than $|\Lambda|$; it increases as the hypothesis size increases. The path visited by the learner goes to the right (as the current hypothesis is specialized) and upwards (as the effective number of constants in the examples increases).

G-Net

A third learner, G-Net (Anglano *et al.*, 1998), based on genetic search, was also considered. G-Net starts with an initial population of candidate hypotheses; these can be viewed as randomly distributed points on a segment of the horizontal line, $L = |\Lambda|$ in the (m, L) plane. The learner navigates on this straight line, moving to the right or to the left, since genetic operators allow candidate hypotheses to be either specialized or generalized. As usual with evolutionary computation-based search, the computational cost of G-Net is significantly higher than that of the other two learners. A smaller number of experiments have therefore been performed with G-Net.

PROGOL

Bottom-up learners like PROGOL and STILL ran out of memory.

STILL

Further experiments have also been done with the relational learners PROGOL (Muggleton, 1995) and STILL (Sebag and Rouveirol, 2000). PROGOL uses background knowledge to tailor the search space and optimize the learning search. However, its clever heuristics make it ill-suited to deal with large artificial problems in the absence of background knowledge and it failed to learn any hypothesis in an acceptable time. As reported by Srinivasan and Muggleton (1995), in the mutagenesis domain some 100 000 seconds are needed to learn on a medium-size problem ($N = 30$, $L = 30$) when there is no background knowledge. STILL uses a bottom-up approach based on the stochastic (uniform or biased) sampling of the matchings between hypotheses and examples. It failed to provide any relevant classifier, owing to the uniform construction of the examples and the lack of any domain bias.

Summarizing the experimental setting, we list the assumptions that have been made; some facilitate the search (these are marked with a $+$) while others make relational learning more difficult (these are marked with a $-$):

- $+$ The training and test sets are equally distributed (100 positive and 100 negative examples), without any noise.
- $+$ All target concepts are conjunctive: a single hypothesis c covers all positive examples and rejects all negative examples.
- $+$ All predicates in the examples are relevant: they all appear in the target concept.
- $-$ All examples have the same size Nm, i.e., the cardinality of a table times the number of predicate symbols in the target concept.

– All predicates have the same number N of tuples (pairs) built on them in every example.

– All variables (predicate arguments) have the same domain Λ of values.

The last three assumptions make learning more difficult because they tend to make the positive and negative examples more similar. Note that, even if the structure of the target concept (m literals involving $n = 4$ variables) were known by the learners, which is obviously not the case, the size of the search space would prevent the solution from being discovered by chance.

10.2 Experimental results

Performances were evaluated with respect to predictive accuracy, concept identification, and computational cost.

To investigate the global impact on relational learning of the phase transition in matching, the three learners mentioned in the previous section were run on the set of selected learning problems. The results were analyzed in the light of the position of the learning problem with respect to the phase transition. Every learner was examined using three specific criteria:

- **Predictive accuracy** The predictive accuracy is commonly measured by the percentage of test examples correctly classified by the hypothesis $\hat{c}$ produced by the learner.[3] It is considered satisfactory iff it is greater than 80% (this threshold value will be discussed later).

- **Concept identification** It must be emphasized that a high predictive accuracy does *not* imply that the learner has discovered the true target concept. The two issues must therefore be distinguished. Identification is considered satisfactory iff the structure of $\hat{c}$ is close to that of the true target concept c, i.e., if $\hat{c}$ is conjunctive, with the same size as c.

- **Computational cost** The computational cost reflects both the total number of candidate hypotheses considered by the learner and the cost of matching each hypothesis on the training set. Typically, the more candidate hypotheses in the phase-transition region, the higher the computational cost.

[3]The predictive accuracy was not evaluated on a test set drawn from the same distribution as the learning set; if the experiments had been doubled, this would have been equivalent to a twofold cross-validation (Dietterich, 1998). We did not double the experiments, because of the huge total computational cost. For the same reason, it was impossible to perform a cross-validation evaluation. Moreover, even though the learning result obtained for any (m, L) is based on a single trial, it can be considered significant to the extent that other trials performed in the same area give similar results.

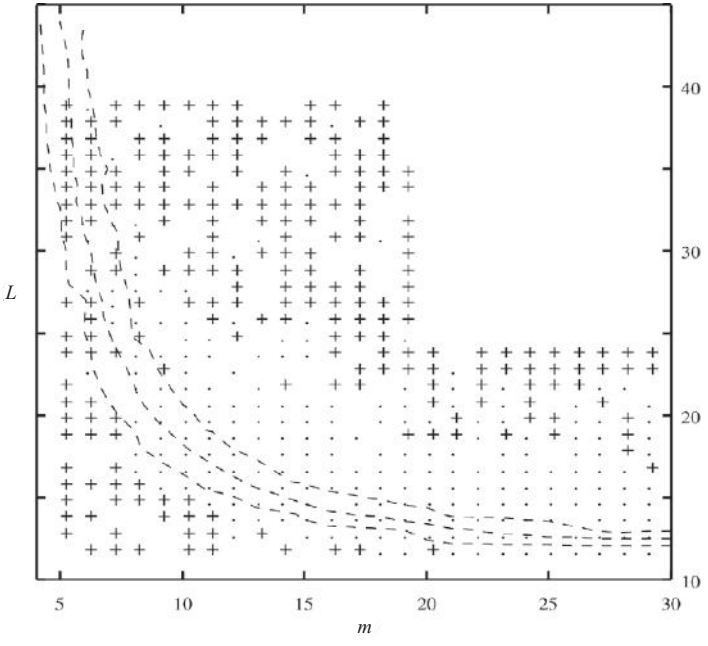


Figure 10.5 FOIL competence map: the *success* and *failure* regions, for $n = 4$ and $N = 100$; L is the number of constants in the model. Plus signs, success (predictive accuracy $\geq 80\%$); dots, failure (predictive accuracy $< 80\%$). The phase-transition region is indicated by the broken curves, corresponding, bottom to top, to the contour plots for $P_{sol} = 90\%$, $P_{sol} = 50\%$, and $P_{sol} = 10\%$, as determined by Giordana and Saitta (2000).

In the following we will review the results obtained by FOIL, SMART+, and G-Net on the artificial relational learning problems constructed as described in the previous section.

10.2.1 Predictive accuracy

Figure 10.5 summarizes the results obtained by FOIL with regard to predictive accuracy. As mentioned earlier, 451 (m, L) pairs were chosen in such a way as to explore significant parts of the YES, NO, and phase transition regions. On each problem, FOIL either succeeds (shown by a plus sign, indicating that the predictive accuracy on the test set is greater than 80%) or fails (shown by a dot.).

A large region emerged, located across the phase transition, where no learner succeeded.

Let us first comment on the significance of these results, with respect to the success threshold and the learning strategy. First, the shape of the failure

region (the *blind spot*) is almost independent of the threshold used to define a failure case (a predictive accuracy on the test set of 80%). In the vast majority of cases the hypotheses $\hat{c}$ learned by FOIL are either very accurate (a predictive accuracy close to 100%) or comparable with a random guess (a predictive accuracy close to 50%). The threshold could thus be any value between 95% and 60% without making any significant difference in the shape of the blind spot.

Second, quite similar results are obtained with FOIL and SMART+, except when SMART+'s beam width is close to the size of the target concept c (meaning that SMART+ performs an almost exhaustive, and much more computationally heavy, search). Complementary experiments with G-Net confirm that the failures should not be blamed on the top-down strategy embedded in FOIL or SMART+. Even though G-Net explored a much larger part of the hypothesis space than FOIL or SMART+ (and was therefore tried for only a few learning problems, situated mostly in the failure region for FOIL), G-Net failed on these problems, too.

Learning succeeds either when the target concept is simple or when it is very complex. Failures occur for concepts of intermediate complexity.

Regarding just the learning performances, it appears that relational learning succeeds in two main cases, either when the target concept is simple (for low values of m), or when the learning problem is far to the right of the phase-transition region.

The first case is hardly surprising; the simpler the target concept, the easier learning should be. Much more unexpected is the fact that learning problems far from the phase transition appear to be easier to solve. In particular, the fact that increasing the number of constants in the application domain *facilitates* relational learning (other things being equal, i.e., for the same target concept size), is counterintuitive. Along the same lines, it is counterintuitive that increasing the size m of the target concept might facilitate relational learning (other things being equal again, i.e., for the same number of constants L). These remarks will be enlarged upon in Section 10.3.

10.2.2 Concept identification

What really happens when FOIL succeeds or fails? Table 10.1 reports the characteristics of the final hypothesis $\hat{c}$ produced by FOIL for a few representative learning problems.

Simple concepts tend to be correctly identified. Complex concepts are approximated by descriptions corresponding to generalizations of the correct one.

The first column indicates the region to which the learning problem belongs. The second column gives the identifier of the problem, which will be referred to in the discussion. Columns 3 and 4 show the "coordinates" of the learning problem i.e., the size m of the target concept c, and the number L of constants in the examples. Columns 5 and 6 refer to the hypothesis $\hat{c}$ learned by FOIL; $\hat{c}$ involves one or several conjunctive hypotheses h_t, iteratively produced by FOIL.

Table 10.1 Hypotheses produced by FOIL for some representative learning problems

Region	Problem	m	L	$\|\hat{c}\|$	$m(h_t)$	Accuracy (%) $\mathcal{E}_L$	Accuracy (%) $\mathcal{E}_T$	CPU time (s)
	Π_0	5	15	1	3	100	100	10.3
Y	Π_1	6	20	1	5	100	99.5	21.4
E	Π_2	7	19	1	7	100	100	52.3
S	Π_3	8	16	1	8	100	100	106.2
	Π_4	9	15	1	9	100	100	69.1
r	Π_5	10	13	1	14	100	99	144.2
e	Π_6	10	16	8	<10−13> 11.75	88	48.5	783.5
g	Π_7	11	13	1	11	100	100	92.2
i	Π_8	11	15	6	<11−16> 13.5	85	53.5	986.2
o	Π_9	12	13	3	<13−15> 14	98.5	83	516.4
n	Π_{10}	13	13	1	13	100	100	455.9
	Π_{11}	14	12	1	13	100	98.5	297.0
	Π_{12}	13	31	13	<1−8> 4.77	90.5	49.5	1317.3
	Π_{13}	15	29	1	6	100	100	185.3
	Π_{14}	15	35	2	<5−7> 6	97.5	84.5	894.6
N	Π_{15}	15	38	1	6	100	99.5	101.5
O	Π_{16}	16	38	3	<5−8> 6.33	97.5	90	1170.6
	Π_{17}	18	24	1	10	100	100	196.4
r	Π_{18}	18	35	1	6	100	100	201.0
e	Π_{19}	19	26	2	<1−8> 4.5	100	98.5	298.4
g	Π_{20}	21	18	8	<1−10> 4.13	81.5	58	1394.9
i	Π_{21}	24	20	1	10	100	99.5	252.3
o	Π_{22}	25	24	1	6	100	99	135.9
n	Π_{23}	27	18	10	<1−13> 5.6	94	72.5	1639.6
	Π_{24}	29	17	1	12	100	99.5	144.9
	Π_{25}	29	23	1	10	100	99.5	720.5
	Π_{26}	29	24	1	9	100	99	618.8
	Π_{27}	6	26	1	6	100	100	82.5
	Π_{28}	6	28	12	<5−11> 8.083 33	91.5	50.5	815.4
	Π_{29}	7	27	11	<5−11> 8.272 73	92	53	1237.0
	Π_{30}	7	28	11	<1−10> 7.636 36	91.5	60.5	1034.2
P	Π_{31}	8	27	1	7	100	100	58.8
T	Π_{32}	11	22	5	<1−12> 3.2	71.5	70.5	851.0
	Π_{33}	11	27	1	8	99	98.5	250.4
r	Π_{34}	13	21	10	<1−11> 4.1	85.5	63	1648.2
e	Π_{35}	13	26	1	9	100	99	476.8
g	Π_{36}	14	20	5	<1−11> 4.8	94	88	722.7
i	Π_{37}	14	24	3	<7−9> 7.666 67	99	92.5	774.0
o	Π_{38}	17	14	8	<13−17> 15	93	46	294.6
n	Π_{39}	17	15	9	<1−13> 5	78.5	66	916.8
	Π_{40}	18	16	8	<1−15> 8.875	91	58.5	404.0
	Π_{41}	19	16	7	<1−12> 8.142 86	83.5	60.5	1268.5
	Π_{42}	26	12	3	<24−25> 24.3333	80	58	361.4

The number of such h_t, denoted $|\hat{c}|$, is given in column 5 (let us recall that the true target concept c is conjunctive, i.e., a correct identification of c implies that $|\hat{c}| = 1$). Maximum, minimum, and average sizes of the conjunctive hypotheses h_t learned by FOIL are displayed in column 6 under the heading $m(h_t)$. These may be compared with the true size m of the target concept (column 3).

The last three columns list the predictive accuracy of $\hat{c}$ on the training and test set and the total CPU time required by FOIL to complete the learning task, as measured in seconds by Botta *et al.* (2003), on a Sparc Enterprise 450.

The learning problems in Table 10.1 can be grouped into three categories.

- *Easy* problems, which are correctly solved. FOIL finds a conjunctive hypothesis $\hat{c}$ that accurately classifies (almost) all training and test examples. Furthermore, $\hat{c}$ is identical to the true concept c or differs by at most one literal. Problems of this type are Π_0–Π_5, Π_7, Π_{10}, Π_{11}, Π_{27}, and Π_{31}. Most easy problems lie in the YES region; others lie in the phase transition for low values of m ($m \approx 6$).

- *Feasible* problems, which are efficiently solved even though the correct target concept is not found. More precisely, FOIL learns a concept $\hat{c}$ which (i) is accurate in prediction (nearly all training and test examples are correctly classified), (ii) consists of a single conjunctive hypothesis, as does the original target concept c, and (iii) shares many literals with c. However, $\hat{c}$ is significantly shorter than c (e.g., $\hat{c}$ involves nine literals versus 29 in c for problem Π_{26}); in many cases, $\hat{c}$ largely overgeneralizes c. Most feasible problems lie in the NO region, rather far from the phase transition. Problems of this kind are Π_{13}, Π_{15}, Π_{17}, Π_{18}, Π_{21}, Π_{22}, Π_{24}–Π_{26}, Π_{33}, and Π_{35}.

- *Hard* problems, which are not solved by FOIL. The learned concept $\hat{c}$ is the disjunction of many conjunctive hypotheses h_t (between six and 15) of various sizes, and each h_t only covers a few training examples. From a learning perspective, FOIL produces overfitting (each h_t behaves well on the training set but its accuracy on the test set is comparable with that of random guessing) related to an apparent small disjunct problem (Holte *et al.*, 1989) even though the true concept is conjunctive. Hard problems lie in the NO region; as opposed to feasible problems, hard problems are close to the phase transition.

These results confirm the fact that predictive accuracy may be only loosely correlated with the discovery of the true concept. Indeed, FOIL succeeds whenever it correctly discovers a single conjunctive concept. It is clear that in real-world problems there is no way of making any difference between feasible and

Table 10.2 Summary of the experiments. Easy and feasible learning problems (Solved pbs.) are distinguished from hard problems (Unsolved pbs.)

Region	No. of pbs.	Percentage of solved pbs.	Average no. of hyps		Avg of solved pbs.	
			solved pbs.	unsolved pbs.	Test acc.	CPU time
YES	46	88.1% (37)	1	6.33	99.61	74.05
NO	195	72.8% (142)	1.27	8.28	99.61	385.43
PT	210	28.1% (59)	1.10	8.18	99.12	238.25
Total	451	52.8% (238)	1.12	7.60	99.45	232.58

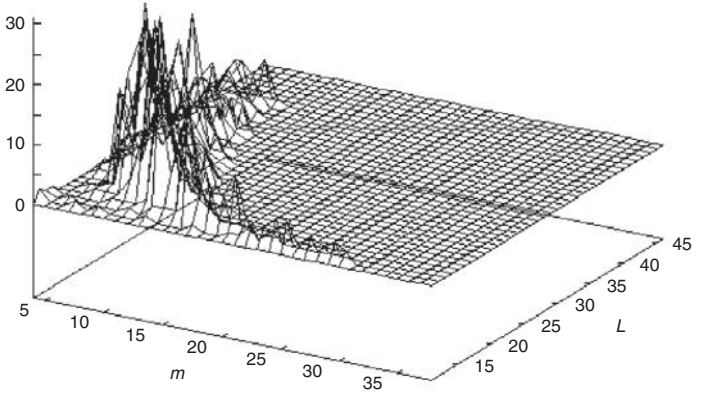


Figure 10.6 Distribution of the number of conjunctive hypotheses h_t (vertical axis) learned by FOIL; they are centered on the phase transition region.

easy problems, since the true concept is unknown. We shall return to this point later.

A first remark concerns the localization of the learning failures. A summary of the average results obtained in the YES, NO, and phase transition regions is reported in Table 10.2. This shows that most hard problems are located near the phase transition; conversely, most problems in the phase transition region are hard. A second remark concerns the localization of the hypotheses h_t learned by FOIL. It is observed that for all learning problems, except the easy problems located in the YES region, *all hypotheses h_t lie in the phase transition region* (Figure 10.6). This is the case no matter whether FOIL discovers one or several

The hypotheses learned by FOIL tend to be in the phase transition region.

conjunctive hypotheses h_t or whether the location of the learning problem lies in the phase transition or in the NO region.

More precisely:

- When the target concept lies in the phase transition region and the problem is easy, FOIL correctly discovers the true concept.
- For feasible learning problems FOIL discovers a generalization of the true concept that lies in the phase transition region.
- For hard problems FOIL retains seemingly random h_t, most of which belong to the phase transition region.

10.2.3 Computational complexity

The computational complexity of the search depends mainly on two factors: the number of hypotheses h_t retained and the average number of their models in an example e, i.e., the number of substitutions θ such that $h_t\theta$ is satisfied in e. For easy problems, a single hypothesis h_t ($\hat{c} \approx c$) is constructed; the computational cost remains low though it increases, as expected, when the average number of models of h_t goes to 1. For feasible problems also, a single hypothesis h_t ($\hat{c}$ most often overgeneralizes c) is constructed. In most cases the computational cost is very low and the average number of models is very high.[4] Finally, in the case of hard problems many hypotheses h_t are constructed and the computational cost is always very high. This might be explained by the fact that most h_t's lie in the phase transition region, and some admit roughly a single model in the examples.

The learning cost is higher for problems in the NO region.

Other things being equal, the learning cost is higher for problems in the NO region. A general cause for this higher complexity is the size of the hypothesis space, which increases exponentially with the number m of literals in c; this causes many more hypotheses to be considered and tested in each learning step. Another cause is that the NO region includes many hard problems (Figure 10.5); for such problems the phase transition is visited again and again as more hypotheses are learned.

10.3 Result interpretation

We will now provide some interpretation of the results reported in the previous section. The discussion focuses on three main questions: why is the learning

[4] A single exception can be seen in Table 10.1: for the learning problem Π_{25}, the average number of models is 1 and the computational cost is high.

search captured by the phase transition region?; when and why does relational learning miss the true target concept?; when and why does relational learning fail to find any accurate approximation of the target concept?

10.3.1 Phase transition as an attractor

In Section 10.2.2 we saw that a large part of the search ends up in the phase transition region, independently of both the location of the true concept and of the used learner. In other words, the phase transition behaves as an *attractor* for the search. Similar results were presented by Botta *et al.* (1999), who showed specifically, using a large set of artificial problems, that FOIL systematically tends to generate concept descriptions located in the phase transition region. In order to understand why, we will take a closer look at what happens during an inductive search by the different learners we have considered.

Learning strategies guided by information gain and minimum description length always end up with a search in the phase transition region.

Being a top-down learner using a hypothesize-and-test strategy, FOIL constructs a series of candidate hypotheses $\{h_1, \ldots, h_i, \ldots, h_t\}$ with increasing specificity. The first hypothesis in the series (just one literal) belongs to the YES region by construction.[5] Then FOIL adds new literals one at a time, moving on the line $L = |\Lambda|$ toward the right and remaining for a while inside the YES region. If the most specific hypothesis h_i that can be built up in the YES region is not satisfactory according to its stop criterion (see below), FOIL possibly comes to visit the phase transition region and so h_{i+1} belongs to it. It might possibly happen that the most specific hypothesis h_j ($j > i$) in the phase transition region is not satisfactory either; FOIL then comes to visit the NO region.

Let us consider the stop criterion used in FOIL. On the one hand the search is stopped when the current hypothesis is *sufficiently correct*, covering no or few negative examples; on the other hand, at each step the current hypothesis is required to be *sufficiently complete*, covering *sufficiently many* positive examples. In the following, the implications of these criteria are discussed for various locations of the target concept c in relation to the phase transition.

Concept c belongs to the phase transition region.

Case 1 The target concept c belongs to the phase transition region

By construction, the target concept c covers with probability 0.5 any example randomly constructed by model RL; therefore, the repair mechanism ensuring equidistribution of the dataset is not employed (Section 10.1.1). We may draw the following conclusions.

[5] Strictly speaking, one should refer to the location of the *learning problem* with respect to the phase transition. A hypothesis with size m does not belong to the phase transition *per se*; this depends upon the number L of constants in the examples. However, for the sake of readability and since L is fixed from the datasets, we shall speak of the location of a hypothesis or of the target concept.

- Since any hypothesis in the YES region almost surely covers any randomly constructed example, it almost surely covers all positive and negative training examples. Therefore the search *cannot stop* in the YES region but must come to visit the phase transition.
- Symmetrically, any hypothesis in the NO region almost surely rejects (does not cover) any random example and will almost surely cover no training example at all. Though these hypotheses are correct they are not admissible since they are insufficiently complete. Therefore the search must stop before reaching the NO region.

From these remarks, it can be seen that FOIL is bound to produce hypotheses h_t lying in the phase transition region.

Case 2 The target concept c belongs to the NO region

In this case, the examples generated by model RL are almost surely negative examples (Section 10.1.1), and half must be turned into positive examples using the **ChangeToPositive** algorithm. It follows that any hypothesis h in the YES region will almost surely cover the negative examples. In addition, it is likely that it also covers the positive ones, because the latter have been constructed in such a way that they are easier to verify; moreover, as any hypothesis generated by the learner uses only the predicates occurring in c, a hypothesis is most likely to be a generalization of c. In any case, any h in the YES region covers at least all negative examples and thus it must be specialized. As a consequence the search cannot stop in the YES region. However, any hypothesis in the NO region will almost surely be correct (i.e., it will cover no negative examples); therefore there is no need for FOIL to engage deeply in the NO region. Hence FOIL is bound to produce hypotheses h_t lying in the phase transition or on the verge of the NO region.

Concept c belongs to the NO region.

Case 3 The target concept c belongs to the YES region

The situation is different here, since there exist correct hypotheses in the YES region, namely the target concept itself and possibly many specializations thereof. Should these hypotheses be discovered (the chances of such a discovery are discussed in the next subsection), the search can stop immediately. In all cases, however, the search must stop before reaching the NO region for the following reason. As c belongs to the YES region, randomly constructed examples are almost surely positive examples (Section 10.1.1). This implies that any hypothesis in the NO region will almost surely reject the positive examples and will therefore be considered insufficiently complete. Again in this case, FOIL is bound to produce hypotheses h_t in the YES or phase transition regions.

In conclusion, FOIL is unlikely to produce hypotheses in the NO region, whatever the location of the target concept c, at least when the negative examples

are uniformly constructed. Most often FOIL will produce hypotheses belonging to the phase transition region, though it might produce a hypothesis in the YES region if c itself belongs to the YES region.

It is worth noting that such a behavior has also been detected in several real-world learning problems (Giordana and Saitta, 2000).

Analogous considerations hold for SMART+, and, more generally, for all top-down learners using a hypothesize-and-test strategy: as maximally general hypotheses are preferred, provided that they are sufficiently discriminant, there is no benefit in searching the NO region. The above argument explains why the *phase transition constitutes an attractor for top-down hypothesize-and-test relational learners*.

Experiments done with G-Net indicate that the same conclusion also holds for a data-driven genetic-based learning search, even though it differs significantly from a top-down search. This finding can be explained as follows. The starting point in a genetic search (the initial population of solutions) consists of random hypotheses distributed on the horizontal line $L = |\Lambda|$ (see Figure 10.4). Then the evolutionary search proceeds by focusing on the fitter hypotheses, where the fitness function favors the most *discriminant and simple* hypotheses. On the one hand, discriminant hypotheses are mostly found close to the phase transition; on the other hand, since simple hypotheses score higher than complex ones, other things being equal, a genetic search will favor hypotheses in the phase transition or on the verge of the YES region. Like FOIL and SMART+, G-Net will most often produce hypotheses in the phase transition region (Giordana and Saitta, 2000).

Discriminant hypotheses of low complexity lie in the phase transition region.

In order to support the claim that most discriminant hypotheses are located in the phase transition region, we conducted another set of experiments. Let us now consider two examples of a given concept c, namely e^+ and e^-, one positive and one negative. Let L_0 be the average number of constants occurring in the two examples. Suppose that the goal is to learn a concept definition φ that covers e^+ and does not cover e^-. Given L_0, model RL generates a set of hypotheses that, paired with e^+ and e^-, constitute matching problems (φ, e^+) and (φ, e^-) corresponding to points on the horizontal line $L = L_0$ in the (m, L) plane. This line intersects the mushy region. We know that, in the NO region, the matching problems (φ, e^+) and (φ, e^-) have very little chance of being satisfied. It would be easy to exclude e^-, but finding a definition for c that covers e^+ may turn out to be a very hard search problem indeed. On the contrary, hypotheses generated in the YES region produce matching problems that are most likely to be verified. Then it is easy to cover e^+ but very difficult to exclude e^-. However, a hypothesis defining a matching problem on the phase transition has probability about 0.5 of verifying any example, so that it should be easier to discriminate between e^+ and e^-.

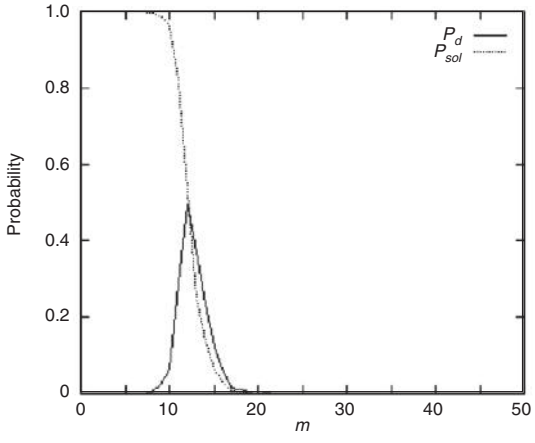


Figure 10.7 Proportion of hypotheses discriminating between two concept instances. For each m value, 1000 formulas were generated corresponding to 2000 matching problems. The largest fraction of discriminant hypotheses corresponds to 50% chance that a solution exists.

On the phase transition edge every formula has about a 50% chance of verifying an example.

In order to test the above conjecture we built up two instances, e_1 and e_2, each with $L = 16$ constants. Moreover, 45 binary predicates were defined, corresponding to relations containing $N = 100$ tuples. Finally, hypotheses with $n = 4$ variables were created according to the procedure used in Section 9.1.3. More precisely, for each value of $m \in [3, 45]$, 1000 formulas were generated, and 86 000 matching problems were defined by pairing each formula with both e_1 and e_2. For each m value the proportion of formulas, P_d, covering exactly one of e_1 and e_2 (i.e., discriminant formulas) was computed and is shown in Figure 10.7. For reference, the graph of the probability of solution P_{sol} is also shown. From the graph for P_d it is clear that the proportion of discriminant formulas reaches its maximum when $P_{sol} = 0.5$, at the phase transition. Therefore, independently of the specific distribution of the concept instances, the part of the hypothesis space that defines matching problems inside the mushy region has a much higher density of acceptable concept definitions than other parts. In conclusion, we formulate the conjecture that any coverage-based induction algorithm will most likely search in this region. The behavior described 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, are as simple as possible, i.e., in the mushy region.

Finally, we analyzed the time evolution of the hypothesis population manipulated by the evolutionary learner G-Net, which was used for the case studies

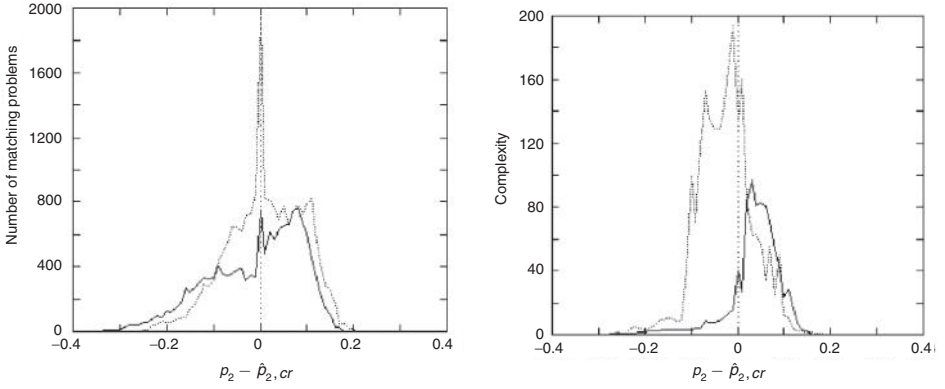


Figure 10.8 Evolution of the population of inductive hypotheses manipulated by G-Net. (a) Distributions of the $p_2 - \hat{p}_{2,cr}$ values for the hypotheses belonging to an initial population (solid line) and to the hypothesis population after 10 000 hypothesis-generation steps (broken line). The concentration of individual hypotheses towards the phase transition clearly emerges. (b) Distributions of the matching complexity for the same populations as in (a). A remarkable increase in the matching complexity appears in the later distribution.

reported in Appendix A. Given a set of examples, Figure 10.8(a) shows the distribution of the variable $p_2 - \hat{p}_{2,cr}$ for matching problems obtained by pairing each example with all the hypotheses belonging to an initial (almost random) population[6] and all the hypotheses belonging to the population reached after 10 000 generation steps. Clearly, as time goes on the hypotheses evolved by G-Net tend to accumulate around the phase transition point, where $p_2 = \hat{p}_{2,cr}$. Figure 10.8(b) shows the corresponding matching complexity, averaged over all problems corresponding to the same $p_2 - \hat{p}_{2,cr}$ value. In conclusion, we expect that a relational learner will tend to explore the phase transition region, being the place where it is more likely to find simple and discriminant hypotheses.

Experiments run with FOIL and G-Net confirm that the phase transition region is an attractor for inductive search.

10.3.2 Correct identification of the target concept

Given that the selected hypotheses are most often close to the phase transition, let us examine why and when these hypotheses might differ from the true target concept c even though c itself belongs to the phase transition.

[6] 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).

When c belongs to the phase transition, two possibilities have been observed (see Table 10.1). If c involves few literals ($m \leq 6$) then it is correctly identified. Otherwise, a number of hypotheses h_t are retained, in which each h_t covers a few positive training examples, and their disjunction $\hat{c}$ performs very poorly on the test set.

The reasons why a top-down learner fails to identify a *long* target concept ($m > 6$) are illustrated with an example. Let us consider the target concept c for problem $\Pi_{8,20}$ ($m = 8$, and $L = 20$), which belongs to the phase transition:

$$c = r_0(x_1, x_2) \wedge r_1(x_2, x_3) \wedge r_2(x_2, x_3), r_3(x_3, x_4) \\ \wedge r_4(x_1, x_4) \wedge r_5(x_1, x_4) \wedge r_6(x_3, x_4) \wedge r_7(x_3, x_4).$$

The top-down search proceeds by greedy optimization of the information gain. The choice of first literal is indeterminate, since any literal covers all positive and negative examples, as explained in Section 10.1.2. This does not penalize the search; in fact, any choice is relevant since all predicates appear in c by construction. All eight specialization paths, corresponding to all possible choices for the first literal, are thus considered in parallel (see Figure 10.9).

Given the first selected literal (say $h_1 = r_0(x_1, x_2)$), the information gain for each literals connected to h_1 is computed, and the literal with the maximum information gain is retained. Unfortunately, it turns out that the best literal according to this criterion (i.e., $r_6(x_3, x_2)$, with gain 270.37) is *incorrect*, i.e., the hypothesis $h_2 = r_0(x_1, x_2) \wedge r_6(x_3, x_2)$ does *not* generalize c; the search cannot recover from this error and it will randomly wander thereafter unless backtrack is allowed (see below).

For this problem, maximization of the information gain appears to be seriously misleading. In all eight specialization paths but one, the first effective specialization step (involving the second literal) fails since FOIL selects incorrect literals (displayed with a broken oblique arrow, together with the corresponding information gain, in Figure 10.9).

When a specialization choice is incorrect, FOIL must either backtrack or end up with an erroneous hypothesis h_t. In order to see the total amount of backtracking needed to find the true target concept c, we correct hypothesis h_2 manually and replace the erroneous literal by the best correct literal (the literal with the maximum information gain such that h_2 generalizes the true target concept). The best correct literal is indicated by a solid vertical arrow in Figure 10.9, together with the corresponding information gain; clearly, the best correct literal often does poorly in terms of information gain. Figure 10.9 depicts all specialization paths.

Unfortunately, it appears that forcing the choice of a correct second literal is not enough; even though h_2 is correct, the selection of the third literal is again misled by the information gain criterion in all branches but one. To pursue the investigation, we iteratively force the choice of the best correct ith literal in all

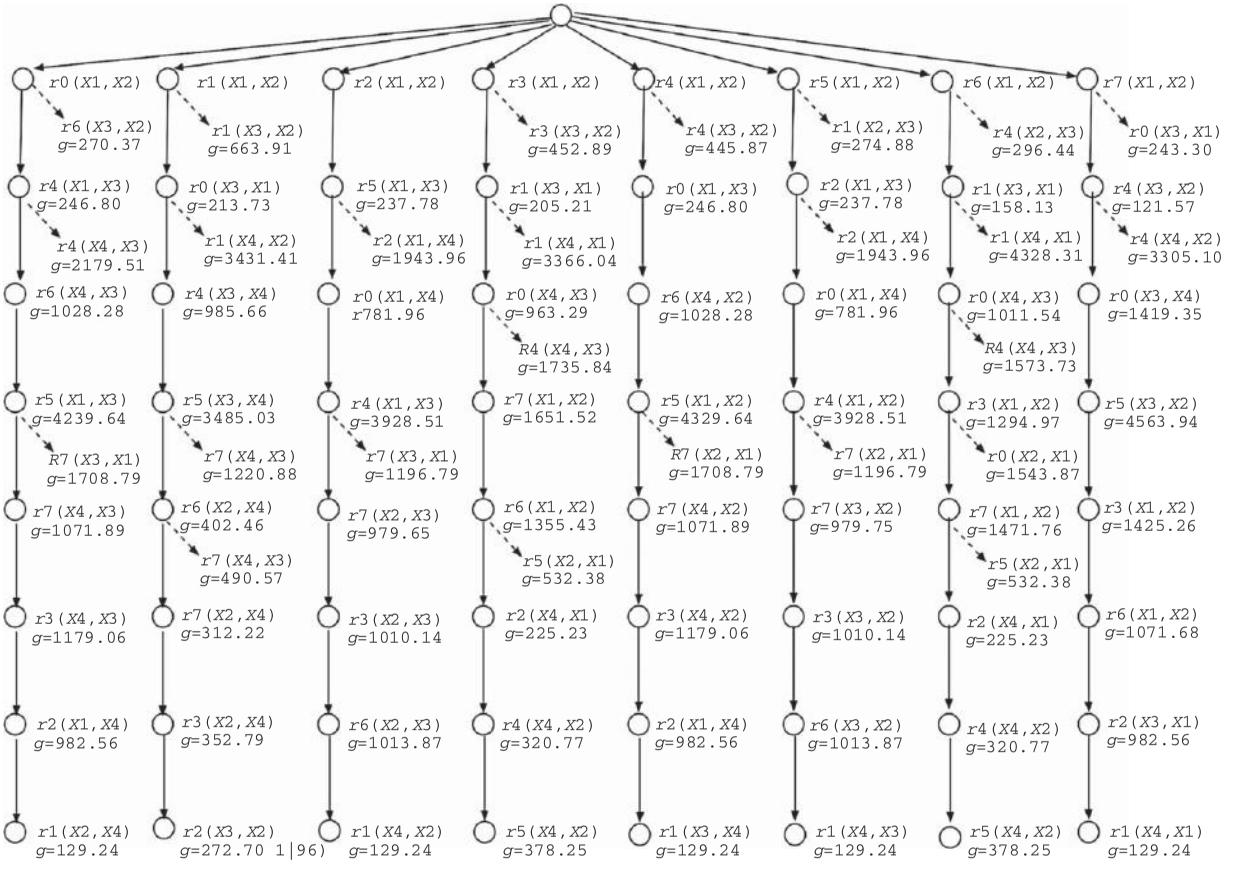


Figure 10.9 Visiting the specialization tree for the problem $\Pi_{8,20}$. All specialization steps "misled" (incorrectly suggested) by information gain maximization are indicated with oblique broken arrows; the incorrect best literal is given with the associated information gain g. In such cases, a backtrack is forced to the best correct literal (indicated with a vertical solid arrow, together with the associated information gain).

cases where the optimal literal with respect to information gain is not correct. All repairs needed are indicated in Figure 10.9.

The above procedure shows that a greedy top-down hypothesize-and-test search is bound to miss the true target concept, as it will not find any error-free specialization path for this learning problem.

10.3.3 Backtrack and domain knowledge

According to Figure 10.9, a huge amount of backtracking would be needed to discover the true target concept from scratch. More precisely, the information gain appears to be reliable only in the late stages of induction and provided that the current hypothesis is correct (the search is "seeded" with four correct literals). In other words, the information gain criterion can be used to transform an educated guess (of the first four literals) into an accurate hypothesis, if the educated guess has reached some critical size (in the particular case of problem $\Pi_{8,20}$ the critical size corresponds to half the size of the true concept). The educated guess to be provided to the learner can be thought of as domain knowledge. The above remarks thus echo the need for strong domain knowledge for learning to proceed, as is generally acknowledged in the inductive logic programming (ILP) literature (Srinivasan *et al.*, 1995; Muggleton, 1992).

In the YES region, the IG measure is not reliable for choosing the literals to be added to the current hypothesis.

The amount of background knowledge needed in order for learning to occur can be evaluated from the critical size m_k of the educated guess, defined as follows. The critical size m_k is the minimal number of literals such that, with probability 0.5, FOIL finds the target concept or a correct generalization thereof (see Section 10.3.4) by refining a m_k-literal guess that generalizes c.

Figure 10.10 shows the critical size $m_k(m, L)$ according to an educated guess for all problems $\Pi_{m,L}$ within or close to the phase transition, obtained as for problem $\Pi_{8,20}$ by systematic backtracking. Figure 10.10 could be thus interpreted as a *reliability map* of the information gain: high values of $m_k(m, L)$ indicate poor reliability.

These empirical limitations of the information gain criterion can be explained within the phase transition paradigm. Let us consider the sequence of hypotheses explored by FOIL (or SMART+). While the current hypothesis h_i belongs to the YES region, it covers any example; the number of substitutions or *models* that satisfy h_i increases exponentially with the number of variables in h_i, regardless of the example label. This hinders the distinction between correct and incorrect literals, as the signal-to-noise ratio is very low. When the search enters the phase transition region, the information gain criterion becomes effective and guides the learner towards one among the many existing discriminant

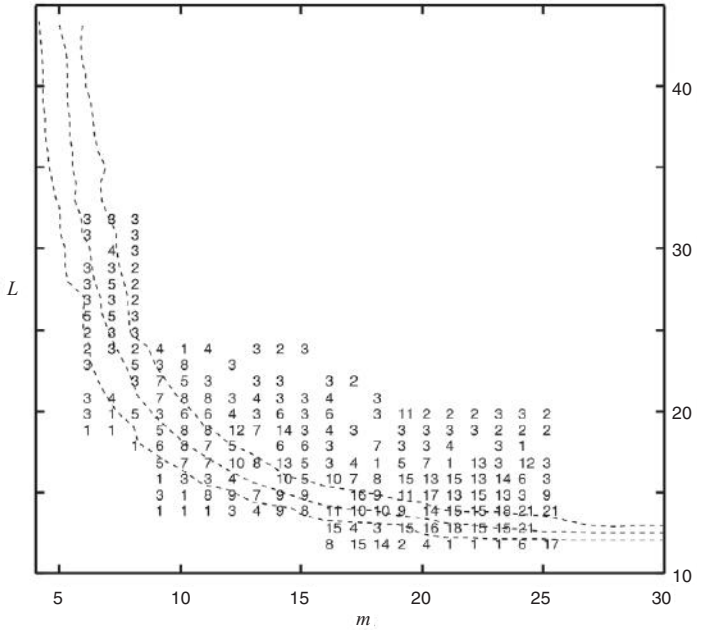


Figure 10.10 Minimum number m_k of correct literals, for all problems $\Pi_{m,L}$ within or close to the phase transition, to be provided before the information gain becomes reliable.

hypotheses. However, the selected discriminant hypothesis may differ significantly from the true target concept, owing to earlier erroneous choices.

To back up these considerations, the average number of substitutions θ (models) such that $h\theta$ is verified in e (where h and e are generated by model RL) and its variance have been measured experimentally. Figure 10.11(a) gives the average number μ of models, for random h and e, as a function of the number m of literals in h, in the cases when h respectively involves 2, 3, and 4 variables; it appears that the number of models decreases very quickly as the phase transition is approached. Figure 10.11(b) shows the standard deviation σ of the number of models, which is very high for all hypotheses in the YES region.

10.3.4 Correct approximation of the target concept

According to the above discussion, the target concept c has hardly any chance of being correctly identified through top-down learning when either its size m or the number L of constants in the application domain are large, which is the case for all problems in the NO region.

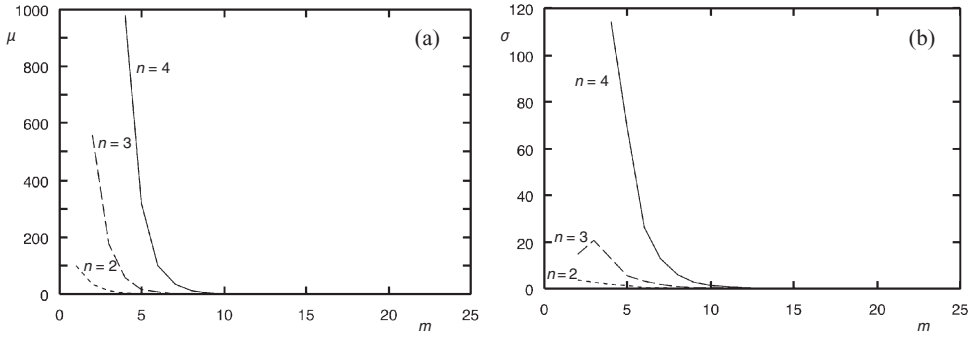


Figure 10.11 (a) The average number μ of models (candidate substitutions θ) such that $h_t\theta$ is verified in e vs. the number m of literals in h_t. (b) The standard deviation σ.

On the contrary, it is observed that FOIL does succeed in finding highly accurate hypotheses (Figure 10.5) for many problems in the NO region, when both the target concept is large and the examples involve many constants (upper-right region, where m and L are large). A closer inspection shows that this is the case when m is more than twice the critical value m_{cr} (where the horizontal line $L = |\Lambda|$, L being the number of constants in the model, meets the phase transition). A tentative explanation for this goes as follows. Let us consider a learning problem in the NO region. As the size m of the target concept increases, so does the amount of modification needed to transform a random example into a positive one (Section 10.1.1). The underlying distributions for the positive and negative examples become more and more different as m increases, which intuitively explains why it becomes easier to separate them.

A top-down search, looking for discriminant concept descriptions, will stop as soon as it finds one in the phase transition region.

More formally, let us consider a generalization φ of the target concept; by construction φ is complete, i.e., it covers all positive examples. However, if φ belongs to the NO region then it almost surely rejects all random examples, and negative examples in particular (the argument closely follows that in Section 10.3.1). All generalizations of c in the NO region are thus almost surely *complete and correct*. Hence, if the learner discovers a generalization φ of the target concept close to the NO region, the learning search stops because φ behaves perfectly on the training set; as φ behaves perfectly on the test set as well, relational learning is deemed to have succeeded. From the standpoint of predictive accuracy, the success of relational learning thus depends on the probability of finding a generalization φ of c on the edge of the phase transition.

Let m and g denote the number of literals of c and φ, respectively. As expected, the number $G(g, m)$ of generalizations of c reaches its maximum for

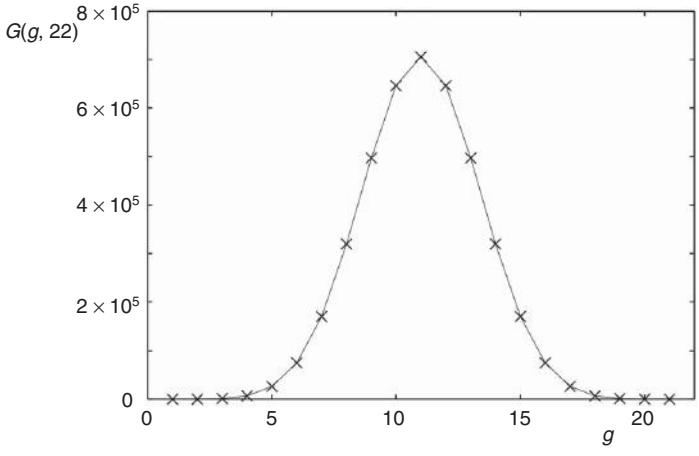


Figure 10.12 The number $G(g, 22)$ of g-literal generalizations of a 22-literal target concept c vs. g.

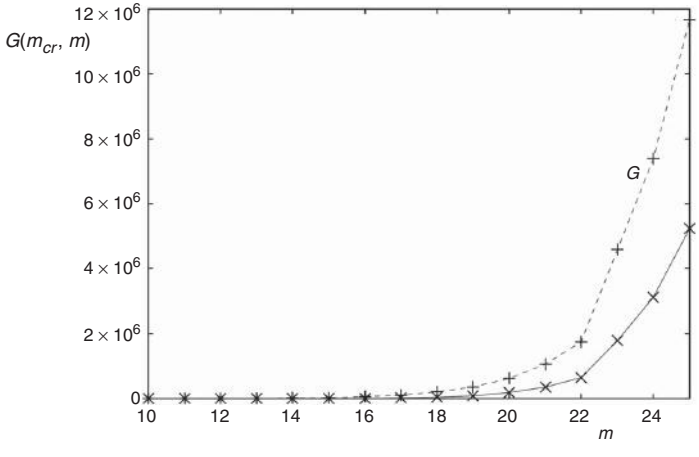


Figure 10.13 The number $G(m_{cr}, m)$ of m_{cr}-literal generalizations of an m-literal target concept c. The number of all g-literal generalizations of c for $g \leq m_{cr}$ is S.

$g = m/2$; Figure 10.12 shows $G(g, m)$ versus g for $m = 22$. Figure 10.13 shows $G(g, m)$ versus m when $g = m_{cr}$; we see that the number of generalizations starts growing very fast as m increases and that half belong to the phase transition region when m is more than twice the critical value m_{cr}.

Both considerations explain why relational learning problems appear to be easier to solve as the size m of the target concept increases *and* is greater than twice the critical value m_{cr}.

10.4 Beyond general-to-specific learning strategies

We have seen that the traditional general-to-specific hypothesize-and-test learning strategies are "misled" by the presence of the phase transition in the covering test. In this section we will investigate other possible strategies, in particular those that are stochastic, knowledge directed, or top-down data-driven. Then we will show that, even if in essence the limits set by the presence of the phase transition cannot be eliminated and no general strategy exists for approaching induction problems of arbitrary complexity, there exist search strategies much more powerful than those used in FOIL, SMART+, and PROGOL.

In previous sections we saw that heuristics relying on the number of models that hypotheses have in the examples (Quinlan, 1990; Botta and Giordana, 1993; Rissanen, 1978) are not reliable in the YES region. The reason is that both incorrect and correct generalizations of the target concept have similar numbers of models in the positive and in the negative examples. In support of this claim, let us consider again Figure 10.11, which shows the average number μ and the standard deviation σ of the model number versus the number of literals in a hypothesis. It is clear that an increase by 1 in the number n of variables induces a large increase in μ and an even larger increase in the standard deviation σ; when $n = 4$ the standard deviation is larger than the number of positive examples in the learning set. Thus, even if we assume that generalizations of the target concept c have at least one more model in the positive examples than in the negative examples, these generalizations cannot be distinguished from a purely random hypothesis. On the contrary, for hypotheses with two or three variables only, μ is much smaller and we may expect that, provided that a correct generalization $\hat{c}$ of c with only two or three variables exists, it should be easier to find it than any generalization having four variables.

FOIL fails when the concept description cannot be approximated by a formula with three variables only.

This conjecture agrees with the fact that many solutions actually generated by FOIL have only three variables even though the target concept is described by a formula with four variables. A more detailed analysis is shown in Figure 10.14, where the line corresponds to the phase transition edge for $n = 4$. In Figure 10.14(a) the plusses indicate the locations of a set of target concepts c with four variables, whose description learned by FOIL also contains four variables. Figure 10.15(a) shows the location of the solutions found by FOIL. Most of these solutions are generalizations of c with four variables, because

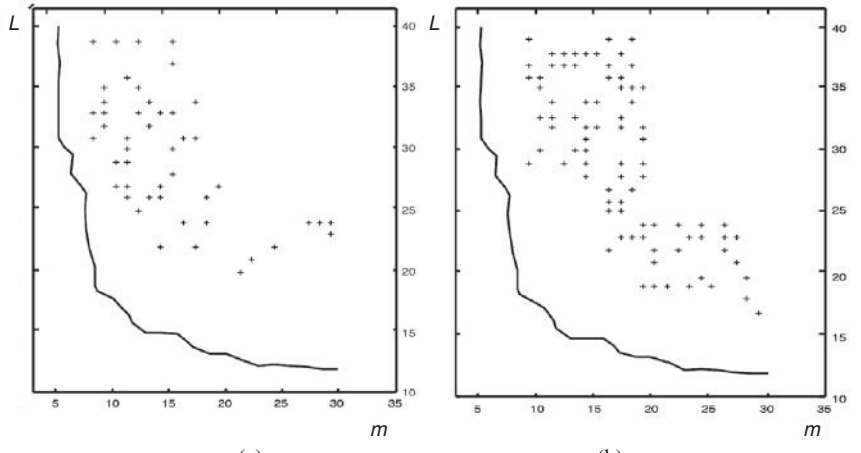


Figure 10.14 Solutions generated by FOIL for problems in the NO region. (a) Problems solved with hypotheses containing four variables; (b) problems solved with hypotheses containing only three variables.

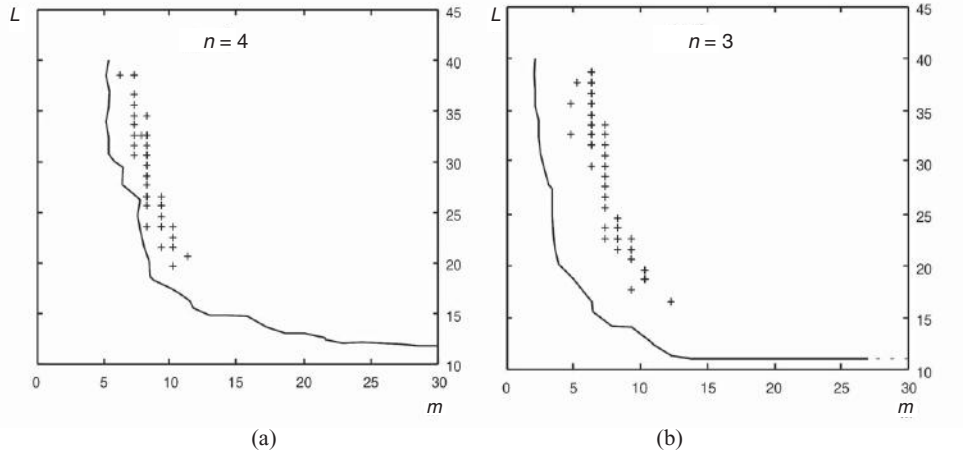


Figure 10.15 Solutions generated by FOIL for problems in the NO region. (a), (b) Locations in the (m, L) plane of the solutions generated by FOIL corresponding to the problems in Figure 10.14(a), (b), respectively.

no acceptable generalizations with three or fewer variables could be found. Figure 10.14(b) shows the locations of a set of target concepts with four variables, whose description as learned by FOIL contains only three variables (see Figure 10.15(b)).

The problems in Figure 10.14(b) are easier to solve than problems in Figure 10.14(a): this result corresponds to the fact that problems in the latter are closer to the phase transition than problems in the former. We observe that the probability that a problem is correctly approximated by a formula with only three variables increases with the number of literals in the target concept c. When c contains more literals, the number of its subformulas that are correct generalizations of c increases as well; then we may expect that at least some contain a number of variables smaller than c's. Moreover, the difficulty of a learning problem increases when L decreases, whereas the critical value m_{cr} increases. When $L < 20$, the only solutions found have three variables.

In conclusion, most learning problems in the NO region have only been solved because approximations with only three variables were sufficient. When more than three variables are required, m_{cr} becomes larger and the available heuristics for top-down search are unable to grow valid inductive hypotheses starting from the YES region. However, we could not find any bottom-up learning algorithm capable of coping with the complexity of working directly inside the NO region.

10.4.1 A stochastic approach

In the absence of a reliable heuristic, stochastic search may be a valid alternative especially if combined with deterministic search. An example of an effective combination of a Las Vegas search algorithm with deterministic search in the *n*-queens problem was given by Brassard and Bratley (1988).

A hybrid approach, combining stochastic search and hill-climbing general-to-specific search, is proposed.

As described in Section 10.3.3, the information gain heuristic is able to guide the inductive search towards a good generalization $\hat{c}$ of a target concept c when an educated hypothesis h_k, containing an appropriate number k of *good* literals, is available.

The algorithm proposed here, which is stated in Figure 10.16, is based on a two-step strategy. The first step creates a hypothesis h_k with a complexity (number of literals) k large enough that h_k lies on the border between the YES region and the mushy region; h_k is the result of random sampling of the hypothesis space. The second step consists in a general-to-specific search, starting from h_k and performing a hill-climbing strategy guided by the information gain heuristic.

If the target concept has a conjunctive description in the hypothesis space $\mathcal{H}$, the Monte Carlo algorithm **SFind** is very likely in the long run to find a c, or at least a good generalization $\hat{c}$ of this c; that is correct on the learning set $\mathcal{S}_L$.

SFind

We would like to compute the complexity of the algorithm **SFind** (see Figure 10.16), assuming that the complexity m and the number of variables n in the concept c are known. By the complexity of the algorithm we mean the minimum number T_ϵ of trials necessary to reach a probability $1 - \epsilon$ of finding $\hat{c}$

Algorithm SFind
Let $\hat{c}_{cur} = \emptyset$
while halt condition does not hold **do**

1. Randomly generate a hypothesis h_k close to the mushy region.
2. Make h_k more specific by following a hill-climbing strategy guided by the information gain. Let $\hat{c}_t$ be the locally best hypothesis found in trial t.
3. **if** $\hat{c}_t$ is better than $\hat{c}_{cur}$ **then** replace $\hat{c}_{cur}$ with $\hat{c}_t$.

end
Return $\hat{c}_t$

Figure 10.16 Stochastic algorithm for searching the space of hypotheses.

(or c, as a specially lucky case). If we choose $k \simeq m_{cr} - 2$, there is strong experimental evidence that a hill-climbing search, guided by the information gain, will find $\hat{c}$ almost surely. The reason is that the search explores the region where the information gain becomes reliable.

Let p_k be the probability that h_k is a subformula of c; then the probability $p_{\hat{c}}^{(t)}$ of finding $\hat{c}$ in no more than t steps is given by the expression

$$p_{\hat{c}}^{(t)} = 1 - (1 - p_k)^t. \tag{10.2}$$

The dependency of the relation (10.2) upon t and p_k is plotted in Figure 10.17. By setting $p_{\hat{c}}^{(t)} = 1 - \epsilon$ in (10.2) and solving with respect to t we obtain

$$T_\epsilon = \frac{\log \epsilon}{\log (1 - p_k)}. \tag{10.3}$$

Given m and n, the probability p_k can be estimated, for a generic number k of literals:

$$p_k = \frac{G(m, n, k)}{S(m, n, k)} \tag{10.4}$$

where $G(m, n, k)$ is the number of subformulas of c with k literals and any number of variables and $S(m, n, k)$ is the total number of formulas in the hypothesis space with k literals.[7] Both $G(m, n, k)$ and $S(m, n, k)$ have been estimated experimentally with an ad hoc algorithm.

[7] The values m and n are those referring to the target concept c.

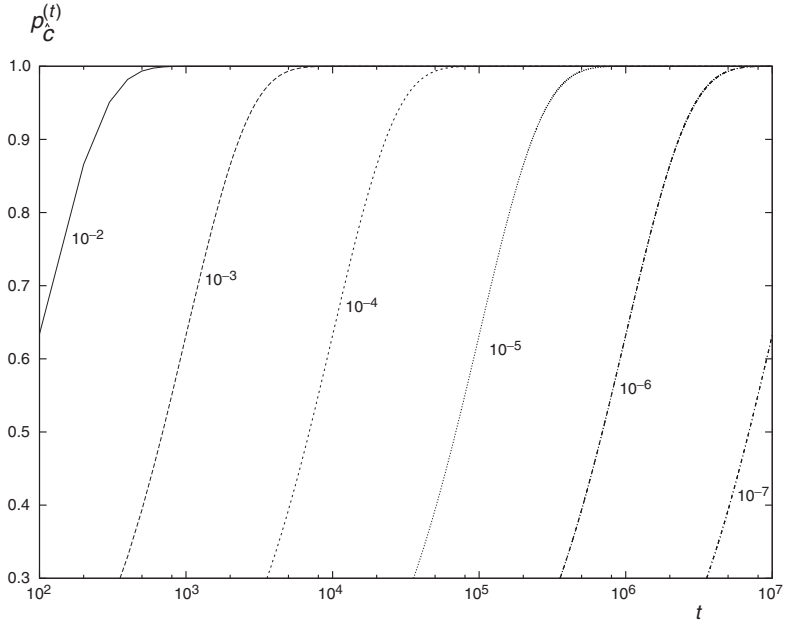


Figure 10.17 Probability that a random searcher will find any subformula h_k of the target concept c vs. the number of trials t, for various values $(10^{-2}$–$10^{-7})$ of p_k. The curve for $p_k = 0.99$ practically concides with the line $p_c^{(t)} = 1$.

The algorithm **SFind** succeeded in solving many learning problems beyond the reach of FOIL.

We note that the value L_{cr} is known when the learning examples are given, because L_{cr} is the average number of constants occurring in them. Table 10.3 reports the results of a set of experiments using Algorithm **SFind**, starting with different values of k until a correct approximation $\hat{c}$ was obtained. Going left to right the columns give:

1, 2. the coordinates (m, L) of the target concept c;

3. the number of literals m_{cr} for a target concept supposedly on the edge of the phase transition;

4. the number k of literals in the stochastically sampled hypothesis;

5. the estimated number of $S(m, 4, k)$ formulas with k literals in units of 10^3 steps;

6. the estimated number $G(m, 4, k)$ of existing generalizations of the target concept, computed by an ad hoc algorithm;

Table 10.3 Results obtained by adding a stochastic search step to the basic hill-climbing strategy. Spaces separate the results obtained for different problems

m	L	m_{cr}	k	$S(m,4,k)$ (10^3)	$G(m,4,k)$ (10^3)	p_k	$T_{0.001}$	$Err\ \%$
7	39	6	4	22.82	0.03	0.001 194 7	5778	100%
7	38	6	4	22.82	0.03	0.001 194 7	5778	100%
7	36	6	4	22.82	0.03	0.001 194 7	5778	100%
7	35	6	4	22.82	0.03	0.001 194 7	5778	100%
7	34	6	4	22.82	0.03	0.001 194 7	5778	100%
7	33	6	4	22.82	0.03	0.001 194 7	5778	100%
7	32	6	4	22.82	0.03	0.001 194 7	5778	47%
			5	190.68	0.02	0.000 099 6	69 379	100%
8	31	6	4	45.64	0.05	0.001 175 8	5871	53%
			5	508.48	0.05	0.000 098 0	70 497	100%
8	30	7	4	45.64	0.05	0.001 175 8	5871	49%
			5	508.48	0.05	0.000 098 0	70 497	100%
8	29	7	4	45.64	0.05	0.001 175 8	5871	51%
			5	508.48	0.05	0.000 098 0	70 497	100%
8	28	7	4	45.64	0.05	0.001 175 8	5871	50%
			5	508.48	0.05	0.000 098 0	70 497	100%
8	27	7	4	45.64	0.05	0.001 175 8	5871	48%
			5	508.48	0.05	0.000 098 0	70 497	100%
9	26	7	4	82.15	0.10	0.001 166 5	5918	49%
			5	1144.08	0.11	0.000 097 2	71 056	100%
9	24	8	4	82.15	0.10	0.001 166 5	5918	50%
			5	1144.08	0.11	0.000 097 2	71 056	100%
10	23	8	4	136.92	0.16	0.001 161 9	5941	51%
			5	2288.16	0.22	0.000 096 8	71 336	48%
			6	24 497.76	0.20	0.000 008 1	856 074	100%
10	22	8	4	136.92	0.16	0.001 161 9	5941	49%
			5	2288.16	0.252	0.000 096 8	71 336	52%
			6	24 497.76	0.20	0.000 008 1	856 074	100%
11	21	9	4	215.16	0.25	0.001 159 7	5953	48%
			5	4194.96	0.41	0.000 096 6	71 476	50%
			6	53 895.07	0.43	0.000 008 1	857 753	100%
12	20	9	4	322.74	0.37	0.001 158 5	5959	51%
			5	7191.36	0.69	0.000 096 5	71 581	50%
			6	107 790.14	0.87	0.000 008 0	858 592	100%
13	19	10	4	466.18	0.715	0.001 158 0	5961	50%
			5	11 685.96	1.13	0.000 096 5	71 581	49%
			6	200 181.70	1.61	0.000 008 0	859 012	49%
			7	2 481 967.49	1.66	0.000 000 7	10 308 184	100%

7. the probability p_k that h_k is a generalization of c;

8. the number of trials $T_{0.001}$ required for a confidence level of 0.999 that a correct approximation will be found by starting from h_k, if it exists;

9. the error rate Err of the best approximation $\hat{c}$ of c on the test set.

10.4.2 Improving the stochastic search algorithm

Expression (10.2) allows one to estimate the number T_ϵ of trials required to reach a confidence level $1-\epsilon$ that the algorithm **SFind** has found an approximation $\hat{c}$, provided that such an approximation exists. The search can be planned as follows.

1. Assume an initial value n for the number of variables in c.

2. Estimate the number T_ϵ of trials necessary to reach the confidence level $1-\epsilon$. If T_ϵ is too high then stop; otherwise go to the next step.

3. Run **SFind** for T_ϵ trials.

4. If the result returned by **SFind** is acceptable then stop; otherwise increase n by 1 and go to Step 2.

We note that the cost for generating and evaluating a hypothesis in the initial random sampling phase is much lower than the cost for the subsequent hill-climbing search. More specifically, the number of hypotheses to be generated and evaluated for the hill-climbing step can be estimated by:

$$\mathcal{O}\big((1+\nu)mn(n-1)\big), \tag{10.5}$$

ν being an integer that has been observed experimentally to range from 0 to 3. In expression (10.5) the term $1+\nu$ is an estimate of the number of literals to be added to a hypothesis h_k in order to cross the phase transition, whereas the term $mn(n-1)$ estimates the number of alternative specializations to be considered at each hill-climbing step. Thus the number of hypotheses to be evaluated in the hill-climbing phase is one or two orders of magnitude larger than the number of stochastic trials.

A second point worth noting is that, assuming that the information gain heuristic is reliable for any hypothesis $h_k \wedge \psi$, it is also likely that h_k itself is scored higher than the average when it is a subformula of c. On the basis of the previous considerations we introduce a new algorithm, which can be more effective than the algorithm **SFind**. Let T_ϵ be the number of trials estimated by

Algorithm T^4

1. Create a set Φ, of cardinality T_ϵ, of hypotheses with k literals.
2. Rank the hypotheses in Φ according to their information gain with respect to the trivial hypothesis $h_0 \equiv true$ (verified on all positive and negative examples).
3. Starting from the top-ranked hypothesis, apply the hill-climbing specialization step to the K best-ranked hypotheses.
4. Return the best description $\hat{c}$.

Figure 10.18 Improved stochastic algorithm.

(10.2) in order to reach a confidence level $1-\epsilon$ in the output of **SFind**. Moreover, let K ($1 \leq K \leq T_\epsilon$) be a user-defined parameter.

$\mathbf{T}^4$ limits the number of hill-climbing steps.

The algorithm T^4 tries to limit the number of hill-climbing steps to the more promising hypotheses, thus reducing the computational complexity. Of course, the parameter K is an arbitrary choice and the confidence level $1-\epsilon$ is guaranteed to be reached only when $K = T_\epsilon$. In practice we have observed that, using values of K that are relatively small ($K = 100$ as against $T_\epsilon = \mathcal{O}(10^5)$), $\mathbf{T}^4$ tends to produce the same results as for $K = T_\varepsilon$.

This algorithm was tested on the set of learning problems shown in Figure 10.5, with the aim of solving those lying in the NO region by following the strategy described above. We started with the minimal hypothesis, with $n = 3$. The parameter k was set to 4, which approximately corresponds to the edge of the phase transition for $n = 3$ and $L \geq 30$. Even though for smaller values of L our strategy predicts larger values of k, it was found that $\mathbf{T}^4$ was also able to find a solution, with the above setting, for many problems where $20 \leq L \leq 30$.

The computational cost of $\mathbf{T}^4$ is comparable with that of FOIL.

Afterwards, more expensive hypotheses were progressively considered, first by increasing k until the value foreseen by the phase transition was reached (or an excessive cost was foreseen) and then by setting $n = 4$. The value $K = 100$ was used everywhere.

The results are given in Figure 10.19. A comparison with Figure 10.5 shows that many problems lying in the blind spot have been solved. In practice, almost all the problems above the phase transition and the line $L = 18$ have been solved with a complexity that is larger than FOIL's but still affordable.

A cluster of 20 Pentium III (800 Mhz) machines was used for the experiments; every problem required a time ranging from few minutes to several hours (an elapsed time comparable to FOIL on a sequential machine). When a problem

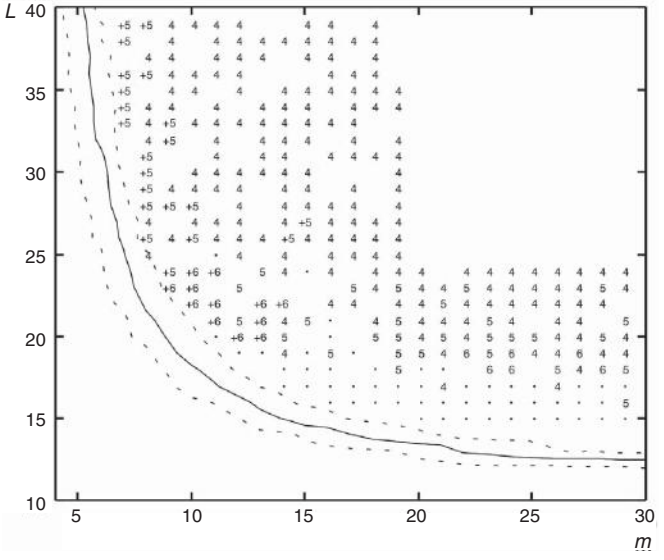


Figure 10.19 Results obtained by the algorithm $\mathbf{T}^4$. The numbers denote the minimum value that had to be assumed for K in order for the learning problem to be solved. When a number is prefixed by a plus it means that $n = 4$ has been assumed; otherwise $n = 3$. A dot means that the problem was not solved.

was not solved, we progressively increased the complexity of the hypothesis by increasing k.

10.5 Comments

Moving from the analysis of the covering test (which is equivalent to a single CSP) to a whole learning problem widens considerably the range and complexity of the phenomena that have been observed, some of which were expected and some not. First, the YES region constitutes a plateau for any top-down searcher exploiting heuristics based on a distinction between the coverage of positive and of negative examples (typically, the information gain). In fact, random hypotheses generated in the YES region almost surely cover any random instance, so that they are unable to distinguish between positive and negative examples, thus making the information gain uninformative. Even when one considers the number of models instead of the number of examples the situation does not improve, because the high variance (see Figure 10.11) masks the difference between the number of models in the positive and negative examples.

However, the YES region is not a local minimum but a *bench* (see Section 6.6), because there are exits towards the phase transition region, where the information gain has higher values than on the plateau. The leftmost points marked with numbers in Figure 10.10 represent, in some way, the border of the plateau. In the NO region the situation would be analogous for a bottom-up learner, but on the negative side: any randomly chosen hypothesis will not cover any randomly chosen example, either positive or negative. However, we have not found any purely bottom-up learning algorithm that is able to attack such computationally demanding learning problems and, hence, the only plateau of interest is that in the YES region.

In an interesting paper, Alphonse and Osmani (2007) showed that the plateau in the YES region may actually be successfully crossed by a top-down searcher that exploits a data-driven strategy, when near-miss negative examples (Winston, 1975) are provided in the learning set. Their experimental results were obtained using a generative model (Alphonse and Osmani, 2008b) derived from model RB (see Section 4.2); this generative model has a guaranteed phase transition for all problem sizes. However, the observed beneficial effects are due rather to the special role of the near misses than to the data-driven strategy; even though the genetic learner G-Net exploits a bidirectional data-driven strategy to learn, it was unable to solve more problems than FOIL or SMART+. These results confirm the intuitive feeling that supplying information to the learner, either in the form of special examples or in the form of the minimum size that the concept should have (see Figure 10.10) may reduce the negative impact of the presence of the phase transition.

The partial solvability of complex problems in the NO region comes as a surprise, as the simplicity of the target concept has always been considered a factor both facilitating learnability and favoring robustness. However, we have to be careful in declaring a success in learning such problems in that the "true" concept was never acquired in these experiments. Not knowing in real learning settings the true concept, good approximations thereof are all we need as long as we do not require perfect classification. Actually, a long concept provides a much larger number of alternative approximations than a short one.

All the investigations presented in this chapter were made with the number of variables n equal to 4. This is roughly the limit one encounters in performing experiments such as those reported here. Most learning problems with five variables could not be handled, owing to the exceedingly high computational resources required. Even though smart algorithms may succeed in reducing the complexity of the single covering test, as was shown in Chapter 9 the sheer number of required in a learning run hinders learning problems from scaling up in size satisfactorily. As mentioned in Chapter 9, this is one reason why the asymptotic behavior with respect to the number of variables is

not of much interest in symbolic machine learning of the type described in this book. In learning statistical relational models, for instance in learning the structure of a Markov logical network, the same issue emerges, because only short clauses (with few variables) can be learned. Only recently some efforts have been devoted to trying to overcome this limitation (Kok and Domingos, 2010; Mihalkova and Mooney, 2007).

A last point worth mentioning is that we have limited ourselves to investigating the emergence of a phase transition in the covering test and its effects on relational learning in general. In all our experiments we kept constant the numbers of positive and negative examples (100 positive and 100 negative examples in both the training and the test set). However, in principle the learning problem *per se* may have some control parameter that, reaching a critical value, determines another phase transition. Actually, this seems to be the case, as Alphonse and Osmani (2009) have shown in relational learning and Rückert *et al.* (2002) have shown in propositional learning. In both cases, the number of negative examples was involved as a control parameter.

11

Phase transitions in grammatical inference

Contents

11.1 Learning grammars

Grammatical inference has been studied since the inception of the theory of formal grammars in the 1960s, in particular to provide a formal framework for language acquisition. Since the pioneering paper of Gold (1967), which introduced the concept of *identification in the limit*, numerous works have been carried out in several scientific communities, including those studying machine learning, pattern recognition, natural language processing, formal language theory, and electronic circuit design. Their goal was to set a theoretical framework for grammatical inference and to design practical learning methods. Recently, these techniques have been used in several application domains such as genomics, natural language processing, and the testing of computer programs.

This chapter provides the necessary fundamental concepts to understand the flavor of this field and reports on a study of generic learning algorithms with regard to the covering test and generalization.

Experimental evidence points again to a phase transition phenomenon, albeit different from those already encountered in this book.

11.1.1 The task of inferring grammars

While so far we have discussed a learning scenario where the task is to extract regularities from sets or collections of (labeled) descriptions, this scenario is far from covering all learning situations. Indeed, much data comes in sequences, and often what one wants to learn is the trend, tendency, or even the rule governing the sequences. Thus, the important relationship is the sequential or temporal organization of the data. This is the case when a child learns the grammar of its native tongue, when a biologist tries to decipher how DNA strands command the fabric of some proteins, or when one wants to guess the oil price next month.

Sequential data

Mathematicians have come up with the concepts of discrete time functions and differential equations to account for time changes. Computer scientists and specialists in signal processing have added other tools, most notably Markov models and grammars, the latter being closely related to abstract machines called *automata*. Markov models are versatile and allow one to represent a large variety of (time) dependencies; however, learning a Markov model from data involves learning first its structure and then the values of its numerous parameters. Grammatical inference is mostly concerned with learning the structure of sequences and is often used as a first step to decide the type of dependencies that are at play. This is why it is relevant to start with the study of grammatical inference.

Formal grammars were originally developed to model natural languages. One key contribution, due to Noam Chomsky (1957), was the definition of a hierarchy of grammars in terms of their generative power and the claim that the syntax of well-formed sentences in natural languages, like sentences in English, could be characterized with respect to one of these grammars (in particular, context-free grammars). Whereas this claim is still controversial, grammars have been used extensively in the analysis and design of computer languages and compilers. Grammars are natural tools for modeling strings of "letters", and as such they have been applied to the study of biological sequences and to many problems in computational molecular biology.

More precisely, Noam Chomsky introduced four types of formal language, which form the *Chomsky hierarchy* of formal languages. These types are distinguished by the types of productions that are permitted in their corresponding grammars.

Types of grammar

Regular Regular grammars[1] are defined by rules of the form $A \to b$ or of the form $A \to bC$.[2]

Context-free Context-free grammars are defined by rules of the form $A \to \alpha$ and are therefore unrestricted in the form that the right-hand side of a rule may take.[3]

Context-sensitive Context-sensitive grammars are defined by rules of the form $\alpha A \beta \to \alpha \gamma \beta$, where γ is not the empty string.

Unrestricted: Unrestricted grammars are identical to context-sensitive grammars except that γ may be the empty string.

Grammatical inference refers to the search for hidden regularities in grammars expressed in strings. For instance, according to Chomsky, language acquisition is mostly the process of discovering the grammar that underlies it.

Take for instance the problem of, given the sequence "`aaabbb`", making a guess at the next element. It is most useful to find the actual generating function (*e.g.*, $\mathtt{a}^\star\mathtt{b}^\star$, meaning a sequence of "`a`"s followed by a sequence of "`b`"s). The questions that concern machine learning specialists are: by which mechanism, i.e., algorithm, can we process the data to find a candidate generating function? Can we obtain such a function from any presentation (any string, in any order)? How fast can we learn? How do we know that we have succeeded? These questions involve three different metaparameters:

1. the *hypothesis space* (or class of functions) considered by the learner;
2. the *protocol of presentation* of the data;
3. the *performance criterion* used to measure success or distance from success.

In grammatical inference the hypothesis space corresponds to grammars that are able to generate or recognize strings, trees, graphs, and other types of structured object. However, influenced by its earlier roots in cognitive science and linguistics, grammatical inference has long been concerned only with learning non-probabilistic grammars. It was indeed thought that probabilities were not fundamentally at play in the learning of languages. Only relatively recently has the learning of probabilistic grammars become an important subject of research. This subject is closely related to that of (hidden) Markov chain models.

[1] Regular grammars produce languages that can be recognized using finite automata.

[2] Upper-case and lower-case roman letters stand respectively for non-terminal and terminal symbols (see Definition 11.5).

[3] Greek letters stand for strings of terminal and/or non-terminal symbols.

From the start, grammatical inference and the field of text learning in general have been much more theoretically oriented than machine learning in its early days. While the latter was at first mainly the theater of many experimental and heuristically oriented studies, there were intense debates and reflections about plausible and controllable protocols in language learning. Again, under the influence of computational linguistics it was thought that negative examples did not play any significant role in the learning of languages. Therefore, *most protocols did not include negative learning examples*, even if some type of interaction with the teacher could be envisioned. In addition, the identification of a language was thought to be the legitimate goal because the role of probabilities in language acquisition was not seen as plausible. More precisely, whereas most of machine learning has yielded to a statistical theory of learning (Vapnik, 1995), in which learning is seen as finding a satisfying hypothesis most of the time (the "probably approximately correct learning" (PAC) setting), *grammatical inference has long retained the goal of exactly identifying the target language*.

Only positive examples

Identification vs. approximation

As a consequence, one requires that candidate hypotheses satisfy the constraints associated with positive instances on the one hand and negative instances on the other hand, i.e., that they "cover", exactly and not in probability, the positive instances and reject the negative instances. This makes the covering test a central component of the learning process, which, at an abstract level, acts as a generate-and-test mechanism. Of course, there is no reason to believe that a target grammar actually exists. It might simply be convenient to hypothesize that the best model of the data, for which one is looking, can be expressed as a grammar. In this case it is possible that no grammar can perfectly label the training data and that one should settle instead for searching for the best, or a good, approximation to the target regularities, whatever they are. One should speak then of grammar induction rather than grammar inference. In any case the covering test still plays a major role in the inductive process.

In addition, as will be shown below, a generality relationship can be imposed upon grammars. This relationship induces a partial ordering in the space of the hypotheses and thus opens the door for algorithms that exploit this structure to search for candidate hypotheses satisfying the constraints imposed by the training examples.

Generality relationship in $\mathcal{H}$

11.1.2 An introductory example

Suppose that some phenomenon in nature or in a man-made device provides us with strings of observations that can be labeled by "$+$" or "$-$". For example, instances of nucleotide sequences could correspond either to introns or to exons. For the sake of simplicity, let us assume that we have received the following sets of instances: $\mathcal{P} = \{aa, a, bb\}$ and $\mathcal{N} = \{ab, b, ba\}$.

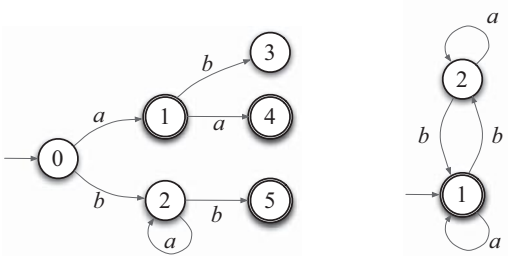


Figure 11.1 Two possible solutions (left) and (right) to the learning problem $\mathcal{P} = \{aa, a, bb\}$ and $\mathcal{N} = \{ab, b, ba\}$. The double rings indicate states that are possibly terminal (i.e., if a sequence ends in this state then it is recognized as belonging to the language).

One could learn a model that would explain the positive instances and reject the negative ones. This model can take the form of a regular grammar in a natural way and, as we will see shortly, this type of grammar can be represented by finite state automata.

There exists an infinity of automata that can explain the above dataset. A learning algorithm could for instance return either the left automaton or the right automaton in Figure 11.1. Both produce the correct label for each training instance. So, which automaton is the best?

In Chapter 5, on machine learning, we showed that a reasonable criterion is that a model should both behave well on the training data and also be "simple". The latter requirement aims at preventing overlearning. Furthermore, a simpler model generally allows for better understanding of the detected underlying regularities. According to this inductive criterion, the automaton at the right is to be preferred.

11.1.3 Automata and grammars

Before studying learning algorithms in grammatical inference and their properties, it is necessary to introduce some notions and terminology that pertain to this field.

Basic notions

Definition 11.1 {***Alphabet***} An *alphabet* Σ is a finite non-empty set of symbols called *letters*.

Alphabets can have as few as two letters, as in the Boolean case, or four letters as in the DNA genetic code or thousands of symbols, as in the Chinese alphabet.

Definition 11.2 {***Strings***} A *string* w over Σ is a finite sequence $w = a_1 \cdots a_n$ of letters. Let $|w|$ denote the length of w. In this case, we have $|w| = |a_1 \cdots a_n| = n$. The empty string is denoted by λ.

The set of all finite strings over the alphabet Σ will be denoted $\Sigma^\star$. Strings can be ordered. Suppose that we have a total order relation over the letters of the alphabet Σ; this is generally called the *alphabetical order*. Then different orders can be defined over $\Sigma^\star$. Without entering into the formal details of their definition, we can single out the *hierarchical order*. With $\Sigma = \{a, b\}$, the first few strings according to that order are $\lambda, a, b, aa, ab, bb, aaa, \ldots$

Given two (possibly empty) strings u and v, we will denote by $u \cdot v$ the concatenation of strings u and v. When the context allows, we will use the notation uv.

Given a string w, x is a substring of w if there are two strings l and r such that $w = lxr$.[4]

Definition 11.3 {***Prefix***} A string u is a *prefix* of another string w if there exists v such that $uv = w$.

EXAMPLE

Given $w = abbaabbaabab$, then $abbaa$ is prefix of w. By contrast, $aabbaa$ is a subsequence but not a prefix of w.

Definition 11.4 {***Language***} A *language* is any set of strings that is a subset of $\Sigma^\star$.

The *complement* of a language L is defined with respect to $\Sigma^\star$: $\bar{L} = \{w \in \Sigma^\star : w \notin L\}$.

The representation of a language by enumeration of the subset of the words of $\Sigma^\star$ that belong to it would be at best cumbersome, especially if the language is not finite. This is why languages are generally represented by grammars of the types defined by Chomsky.

A *grammar* is a mathematical object associated with an algorithmic process that can generate a language.

[4] With this definition, x is always a substring of itself.

Definition 11.5 {***Grammar***} A *grammar* is a quadruplet $G = (N, \Sigma, P, S)$, where:

- N is an alphabet of all the *non-terminal symbols* of G;[5]
- Σ is the *terminal alphabet* of G and is distinct from N. We set $V = N \cup \Sigma$;
- $P \subseteq (V^\star N^+ V^\star \times V^\star)$ is a finite set of *production rules*;
- $S \in N$ is an *axiom* of G.

A *production rule P* is written as $\alpha \longrightarrow \beta$, with $\beta \in V^\star$ and $\alpha \in V^\star N^+ V^\star$, which means that α contains at least one non-terminal symbol.

Definition 11.6 {***Word generated by a grammar***} A word $v \in \Sigma^\star$ is said to be *generated* by a grammar G when it can be generated from the axiom S of G.

The *language generated by the grammar* G is the set of all the words in $\Sigma^\star$ that can be generated by G. We denote it by $L(G)$.

Here are two examples. When there is no ambiguity, one can simplify the description of the grammar by providing its rules only and by writing in a line all the rules with the same left-hand side.

—— EXAMPLE ——

The grammar defined by $N = \{S\}$, $\Sigma = \{a, b, c\}$ and $P = \{(S \longrightarrow aSb), (S \longrightarrow \epsilon)\}$ can be written as

$$S \longrightarrow aSb \mid \epsilon$$

This grammar generates the language $\{a^n b^n \mid n \geq 0\}$. Indeed, one can see, taking an example, that its axiom S allows the derivation of the word $aaabbb$ in four steps: three applications of the rule $S \longrightarrow aSb$ and one of the rule $S \longrightarrow \epsilon$, giving successive steps

$$S \quad aSb \quad aaSbb \quad aaaSbbb \quad aaabbb.$$

Chomsky noticed that the representation of languages using grammars allows the definition of language types in terms of the production rules that generate the languages. This classification introduces a strict hierarchy among the classes of languages. The *regular grammars* are the simplest. They are also called *rational languages* because they form the smallest family of languages

[5] Non-terminal symbols are not observed in the strings of a language (see Definition 11.4) but are used as place-holders or auxiliary symbols in the derivation of sentences.

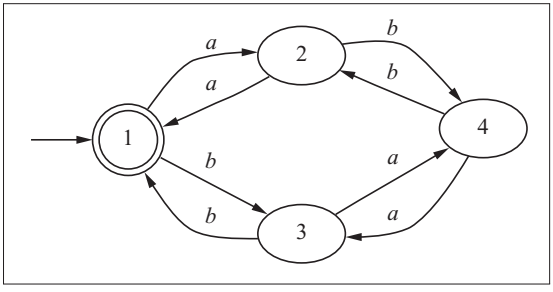


Figure 11.2 A finite state automaton accepting strings composed of an even number of a's and an even number of b's.

on $\Sigma^\star$ that is closed by the rational functions that are the union, the product of concatenations, and Kleene's iteration operation $\star$.

Regular grammars

A *regular grammar* is defined by $A \longrightarrow wB$ or $A \longrightarrow w$, with $w \in \Sigma^*$, $A \in N$, and $B \in N$.

A language that can be generated by a regular grammar is said to be a *regular language*. A classical result of the theory of languages is the following.

Theorem 11.1 *Any regular language can be generated by a finite automaton. Conversely, any finite automaton generates a regular language.*

Graphical representation of finite state machines

Automata can be seen as a kind of graphical representation of regular grammars, where the non-terminal states correspond to non-terminal symbols and the transitions or edges to the derivation rules. By convention, a terminal state is represented with a double circle and the initial state is indicated by an entering arrow.

EXAMPLE

The grammar defined by $N = \{1, 2, 3, 4\}$, $\Sigma = \{a, b\}$ and $P = \{(1 \longrightarrow a2), (1 \longrightarrow b3), (1 \longrightarrow \epsilon), (2 \longrightarrow a1), (2 \longrightarrow b4), (3 \longrightarrow a4), (3 \longrightarrow b1), (4 \longrightarrow a3), (4 \longrightarrow b2)\}$ is strictly equivalent to the finite state automaton of Figure 11.2.[6] Strings that are accepted are composed of an even number of a's and an even number of b's.

This grammar can be rewritten more simply as:

$$1 \longrightarrow a2 \mid b3 \mid \epsilon, \qquad 2 \longrightarrow a1 \mid b4,$$
$$3 \longrightarrow a4 \mid b1, \qquad 4 \longrightarrow a3 \mid b2$$

[6] Here, the axiom is not denoted S, but 1.

Words accepted by an automaton

The words accepted by the automaton are those for which there exists a path from the initial state to a terminal state through the transitions dictated by the symbols of the words of the language.

Finite automata

Automata are finite state machines that can recognize strings. They correspond to a simplified and limited version of Turing machines. A string is provided on the input tape; it is then read from left to right and, at each step, the next state of the system is chosen depending on the previous state and the letter or symbol that is read. The automata is deterministic if only one action is possible at each step. Deterministic finite automata are usually preferred because they are simpler to manipulate and lead to more efficient parsing and also because a number of theoretical results apply only to them. However, nondeterminism may be better suited for modeling certain phenomena and could also be a partial solution to the difficulties one has when facing noisy data.

As claimed by Theorem 11.1, finite automata are equivalent to regular grammars. We are now going to define them more precisely.

Definition 11.7 {***Finite automata***} A *finite automaton* is a quintuplet $(Q, \Sigma, \delta, q_0, F)$, where Q is a finite set of *states*, Σ is a *finite alphabet*, δ is a *transition function*, i.e., a function from $Q \times \Sigma$ *to* 2^Q, $Q_0 \in Q$ is the subset of *initial states* and $F \in Q$ is the subset of *final states*, also known as *accepting states*.

First, we consider deterministic finite state automata, then the nondeterministic case.

Deterministic finite state automata

Definition 11.8 {***Deterministic finite state automata (DFA)***} If, for every state $q \in Q$ and for every letter $a \in \Sigma$, the transition function $\delta(q, a)$ contains at most (respectively exactly) one element and if Q_0 contains only one element q_0, the automaton A is said to be *deterministic* (respectively *complete*).

In what follows we will use the abbreviations DFA for deterministic finite state automata and NFA for non-deterministic finite state automata.

EXAMPLE

The automaton of Figure 11.2 is a DFA. Another example of a finite state automaton is given in Figure 11.3. It contains five states, $Q = \{0, 1, 2, 3, 4\}$. It is defined over the two-letter alphabet $\Sigma = \{a, b\}$. The initial states are 0 and 5, $Q_0 = \{0, 5\}$, and the states 3 and 4 are final, $F = \{3, 4\}$.

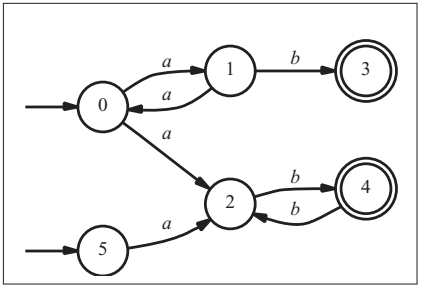


Figure 11.3 A non deterministic finite state automaton.

This is an NFA since there are two edges labeled with the letter a from state 0 and since there are two initial states. Here, $\delta(0, a) = \{1, 2\}$.

Furthermore, this automaton is not complete since the transition function δ is not everywhere defined. For instance, $\delta(5, b)$ is not defined.

Language accepted by a finite automaton

A language $L(A)$ accepted by an automaton A is the set of all strings that are accepted by A, i.e., for which there exists a sequence of states that are "excited" from an initial state to a final state when one uses the transition function on the successive letters of the string. For instance, the sequence of states $(0, 1, 3)$ is associated with an acceptation of the sequence aab in the automaton of Figure 11.3.

It is sometime possible to remove some states from an automaton without changing its accepted language. This leads to the definition of a minimal accepting automaton.

Definition 11.9 $\{$***Minimal deterministic finite automata***$\}$ For any regular language L there exists a DFA $A(L)$ that generates L and has a minimal number of states; $A(L)$ is called the *minimal deterministic automaton* or *canonical automaton* associated with L. It can be proved that this automaton is unique.

EXAMPLE

The automaton of Figure 11.4 *accepts* a language composed of the strings that start with an odd number of a's followed by an odd number of b's (which corresponds to the regular expression $L = a(aa)^\star b(bb)^\star$). This is the same language as that accepted by the automaton of Figure 11.3.

There exists no automaton that contains fewer states than this and that accepts the language L. This is therefore the canonical automaton corresponding to language L.

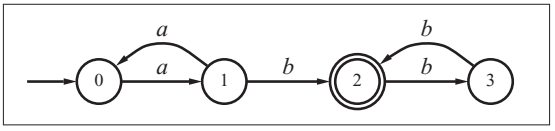


Figure 11.4 Canonical automaton of the language defined by the regular expression $L = a(aa)^*b(bb)^*$.

Nondeterministic finite state Automata

While most published works deal with the implications of deterministic finite state automata, it is the case that nondeterministic finite state automata (NFA) have interesting properties also. For instance, they may offer compact representations of some regular languages. It may thus happen that an NFA needs an exponentially smaller number of states than its corresponding deterministic finite state automaton.

In addition, the use of NFA offers different ways of expressing the same language, thus allowing one to choose the representation most adapted to one's needs. This is why, in many domains, for example in genomics, NFA are used to represent biological regularities of interest even though there are fewer learning algorithms for NFA.

In the following, we will be looking at both DFA and NFA.

11.1.4 Learning automata

There exists a complete one-to-one mapping between the four types of grammars in the hierarchy of Chomsky and the four types of automata.

Grammar	Regular	Context-free	Context-sensitive	Unrestricted[7]
Automata	Finite state automata	Pushdown automata	Bounded-tape Turing machines	Turing machines

Theoretical works on grammatical inference show that the class of regular languages is the most general class of languages that is identifiable in the limit in polynomial time. In practice this means that regular languages are sufficiently expressive for a wide range of applications while remaining effectively learnable.

[7] Also known as recursively enumerable.

This is the reason why the learning of regular languages has been the subject of numerous studies.

11.2 Grammatical inference by generalization

We now turn to a hypothesis space consisting of grammars or finite automata. We will show that there exists a partial ordering that can be defined over such a space that will allow us to use learning methods that exploit this ordering.

11.2.1 The space of finite automata

In the hypothesis spaces that we have considered so far, the crucial structure that permits a well-guided search for good hypotheses was induced by the relation of inclusion in the space of examples. Thus, in Chapter 5, it was said that one hypothesis, h_i, is more general than another, h_j, if the set of instances that hypothesis h_i labels as positive (i.e., covers) includes the set of instances covered by h_j. While this induced ordering was immediate in the case of propositional representations, it was more involved in the case of relational concepts and entailed some special care.

In the field of grammatical inference, likewise, one wants to find a *description* of the target language (or of some language close to it) that is not in extension, by which we mean not in the form of the set of all accepted sentences. As we have seen, a favored description takes the form of automata. The question is therefore how to induce a partial ordering on the space of finite automata from the ordering associated with the inclusion relation defined over the space of sentences.

Derived automata

A central operation over automata is the partitioning of its states. Recall that a *partition* π of a set S is a set of subsets of S such that these subsets are not null, are disjoint by pairs, and their union is equal to S. When defined over the set of states of an automaton $\mathcal{A}$, a partition determines a *quotient automaton* $\mathcal{A}/\pi$ where all states belonging to one subset in the partition are merged together (see an example of the Figure 11.5).

One fundamental property of the state-merging operation is that if an automaton $\mathcal{A}/\pi_j$ derives from an automaton $\mathcal{A}/\pi_i$ through state-merging operations then the language accepted by $\mathcal{A}/\pi_i$ is included in the language accepted by $\mathcal{A}/\pi_j$. In other words, *the operation of merging states of a finite state automaton induces a generalization of the languages accepted*.

State merging induces a generalization in the space of automata.

Thanks to this property it is possible to build all the automata that can be derived from a starting automaton $\mathcal{A}$ by enumerating all the partitions of its states.

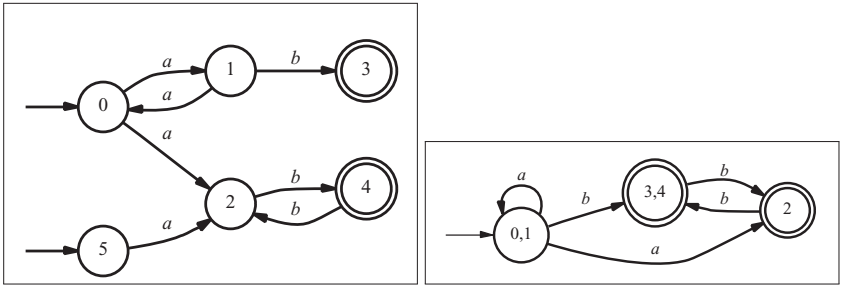


Figure 11.5 Left, a nondeterministic finite state automaton $\mathcal{A}$. Right, the quotient automaton $\mathcal{A}/\pi_1$, where $\pi_1 = \{\{0, 1, 5\}, \{2\}, \{3, 4\}\}$.

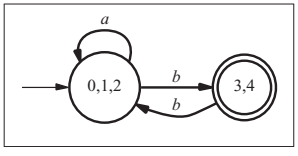


Figure 11.6 The quotient automaton $\mathcal{A}/\pi_2$ derived from $\mathcal{A}/\pi_1$ with $\pi_2 = \{\{0, 1, 2, 5\}, \{3, 4\}\}$.

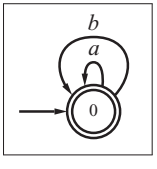


Figure 11.7 The universal automaton over the alphabet $\Sigma = \{a, b\}$.

Furthermore, as we have seen, there exists a partial ordering on the space thus defined that is consistent with the inclusion relationship between languages.

The automaton where all states are merged into a single state is called the *universal automaton* (UA). It accepts all strings defined over a given alphabet.

For instance, the universal automaton defined over the alphabet $\{a, b\}$ is represented in Figure 11.7.

A lattice over the space of automata

From the above property it can be inferred that the hypothesis space can be endowed with a lattice structure. Indeed, it can be proved that the set of the automata derived from a given automaton $\mathcal{A}$, partially ordered by the derivation

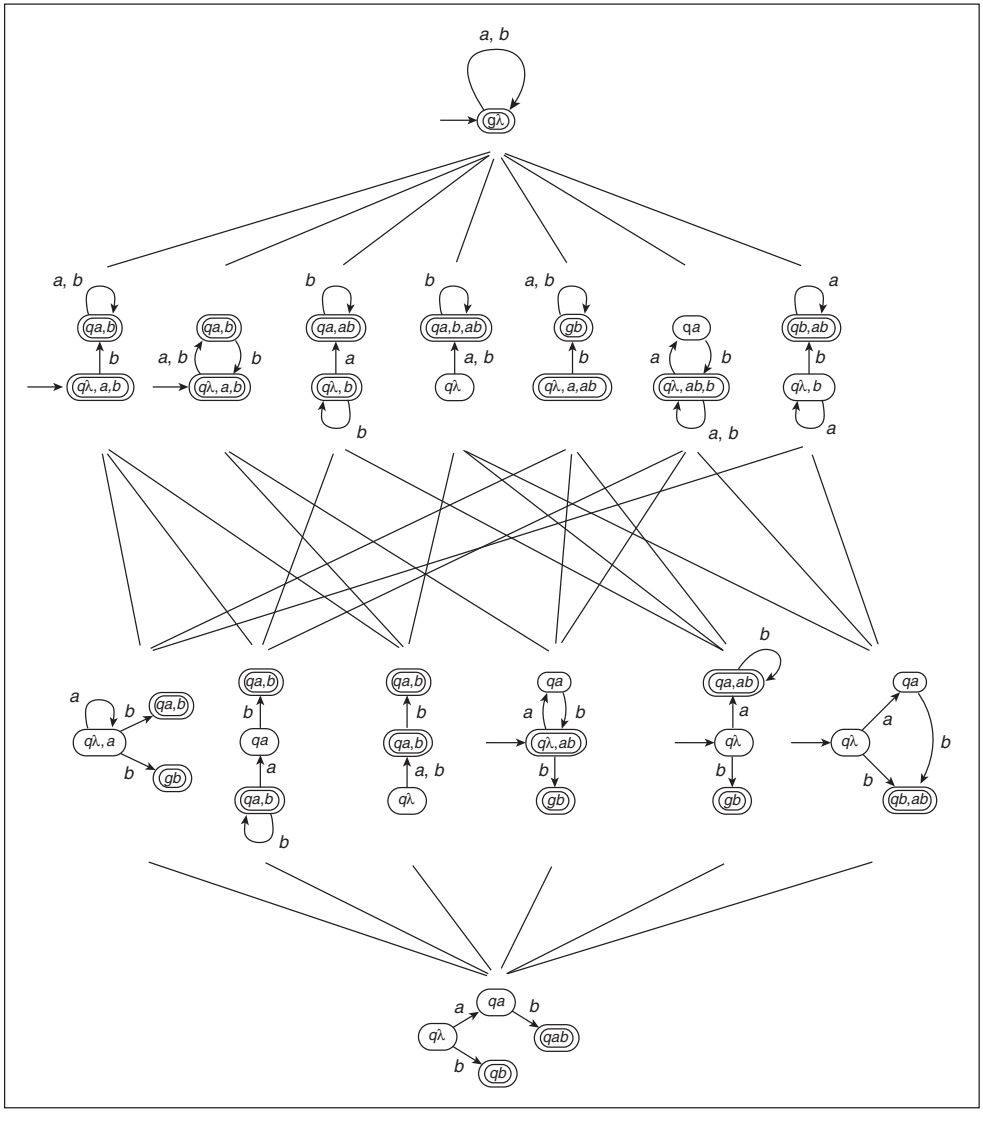


Figure 11.8 The lattice for the derived automata built on $MCA(\{\mathrm{a}, \mathrm{ab}\})$ (courtesy of Colin de la Higuera).

operation, is a lattice; the automaton $\mathcal{A}$ and the universal automaton (UA) are respectively its minimal and maximal elements. This lattice is denoted $Lat(\mathcal{A})$. Figure 11.8 shows *Lat* ($\mathcal{A}$) when the automaton $\mathcal{A}$ is the maximum canonical automaton (*MCA*) of a set of strings, which is a star-shaped NFA with one branch per string.

11.2.2 A structure for the space of finite automata

Up to now we have seen how it is possible to endow the space of finite automata with a *partial order associated with the generality relationship* between automata. An automaton derived from another accepts at least all the strings accepted by the latter. It therefore recognizes a more general language.

However, we have not yet introduced the notion of a *training sample* or *training set* from which we want an automaton to learn.

Given a sample $\mathcal{S}_L = \mathcal{P} \cup \mathcal{N}$, where $\mathcal{P}$ contains the *positive examples* and $\mathcal{N}$ contains the *negative examples*, the empirical risk-minimization principle dictates that an optimal hypothesis is one that minimizes the error on the sample. Actually, as shown in Chapter 5, we should take into account the complexity of the hypothesis space in order to obey a regularized inductive principle. In grammatical inference we usually seek a consistent hypothesis, one that does not make any error on the training set. Of course, this assumes that the data is not suspected of being noisy. We will see how regularization is taken care of by generic learning methods.

The inductive criterion

Structural completeness

The number of automata consistent with a given (not conflicting[8]) training sample is infinite. Additional knowledge or bias must therefore be provided to make learning possible. For instance, suppose that the alphabet contains the letters a, b, and c but that only the letters a and b are seen in the positive examples. Then, unless we are told otherwise, there should be no reason to introduce the letter c in the candidate automata. In other words, we will assume that the training sample is sufficiently representative of the language to be learned. In concrete terms this means that every component of the target automaton is exercised in the recognition of at least one learning example. Most standard grammatical inference systems that have been devised follow this assumption since it confers desirable properties on the search space.

Definition 11.10 {***Structural completeness***} A set $\mathcal{P}$ is said to be *structurally complete* with respect to a DFA A if $\mathcal{P}$ covers each transition of A and uses every element of the set of final states of A as an accepting state.

Structural completeness can be extended, with care, to the nondeterministic setting. We do not detail this here. The interested reader can consult de la Higuera (2010) with profit.

We will therefore search only for finite automata for which the training sample is structurally complete.

[8] That is, where $\mathcal{P} \cap \mathcal{N} = \emptyset$.

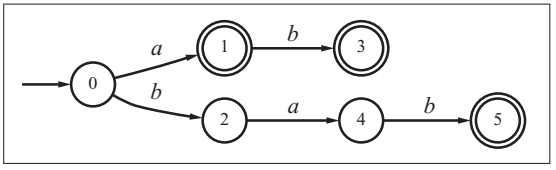


Figure 11.9 $PTA(\{a, ab, bab\})$.

As we have seen above, the space of automata can be endowed with a generality-based partial ordering. Furthermore, given a positive training sample $\mathcal{P}$, the search space, under the structural completeness hypothesis, has the following properties.

Theorem 11.2 *Every DFA in the lattice partition $Lat(MCA(\mathcal{P}))$ is structurally complete for $\mathcal{P}$. Likewise, every NFA in the lattice partition $Lat(MCA(\mathcal{P}))$ is structurally complete for $\mathcal{P}$.*

Conversely, we have the following.

Theorem 11.3 *Every DFA that is structurally complete for $\mathcal{P}$ is in the lattice partition $Lat(MCA(\mathcal{P}))$. Likewise, every NFA that is structurally complete for $\mathcal{P}$ is in the lattice partition $Lat(MCA(\mathcal{P}))$.*

If, furthermore, we are interested in finding the smallest deterministic finite state automaton accepting $\mathcal{P}$, then we can reduce the search space.

First, let us define the prefix tree acceptor (PTA) associated with a positive training sample.

Definition 11.11 $\{$***Prefix tree acceptor (PTA)***$\}$ A *prefix tree acceptor* (PTA) is a tree-like DFA built from the learning sample by taking all the prefixes in the sample as states and constructing the smallest DFA which is a tree such that $\forall q \in Q, |\{q' : \delta(q', a) = q\}| \leq 1$, i.e., that each state has at most one ancestor.

EXAMPLE

The automaton of Figure 11.9 is the prefix tree acceptor for the positive sample $\mathcal{P} = \{a, ab, bab\}$.

Thus, we have the following property.

Theorem 11.4 *The smallest DFA consistent with a sample $\mathcal{S} = \mathcal{P} \cup \mathcal{N}$ is in $Lat(PTA(\mathcal{P}))$.*

In addition, since it can be shown that $PTA(\mathcal{P})$ derives from $MCA(\mathcal{P})$, we have $Lat(PTA(\mathcal{P})) \subseteq Lat(MCA(\mathcal{P}))$ and therefore searching a solution in $Lat(PTA(\mathcal{P}))$ is generally more efficient.

The generic approach

This property immediately suggests a learning approach for a DFA. The principle is to start the exploration of the search space $Lat(PTA(\mathcal{P}))$ from $PTA(\mathcal{P})$ and then to explore it using the state-merging operator. Overgeneralization is prevented by using $\mathcal{N}$. No candidate automaton should accept a string from $\mathcal{N}$.

This strategy is at the core of many learning algorithms, such as RPNI, that are widely used.

A difficulty for context-free grammars

It must be noted that the crucial "more specific than" relationship is undecidable for context-free grammars. This of course precludes the use of learning algorithms guided by this well-informed relationship and explains, in part, why so little work has been devoted to learning algorithms for context-free grammars (see, however, Lehn and Ball (1987)).

11.3 A phase transition in learning automata?

We saw in Chapters 9 and 10 that an important step in the study of learning was the realization that learning could be cast as a constraint satisfaction problem. In the case of concept learning, one is often interested in discovering a hypothesis that is consistent with the training data. It should make no errors, which means that it should predict as positive the positive examples and as negative the negative examples. At an abstract level the problem amounts to checking that there exists at least one hypothesis that can satisfy the constraints imposed both by the positive and by the negative training instances.

While in the SAT domain the question raised was the importance of the value of k in the k-SAT problem, in machine learning and in artificial intelligence in general, the focus is rather on the expressiveness of the hypothesis language. It has been known since the fundamental work of Brachman and Levesque (2004) that the tractability of deduction and the expressiveness of the supporting language are tied by a trade-off: the more expressive is the language, and therefore its ability to leave things unspecified, the more intractable is deduction using this language.

Expressiveness and phase transitions

In machine learning the pioneering work on the covering test in first-order logic (see Chapter 9) and the corresponding display of a phase transition phenomenon suggested that languages as expressive as first-order logic were likely to be conducive to similar abrupt transitions in the covering test. This, in turn, is indicative of potential severe difficulties for learning concepts expressed using these languages.

In the same way that in SAT we raised the question whether there is a boundary between the uneventful 2-SAT problem and the phase-transition-prone 3-SAT problem, the question whether there is some kind of threshold in the language expressiveness between zeroth-order logic (propositional logic) and first-order logic was, and still is, open in machine learning.

From in this perspective the study of automata learning seemed appropriate. Indeed, while grammars, and especially regular grammars, are less expressive than first-order logic they are more expressive than propositional logic. Although language expressiveness is not just a one-dimensional property, it was felt that it might be illuminating to study the covering test in the case of automata learning.

Should we then expect a phase transition in learning automata? No theoretical argument then existed. Cornuéjols, Pernot, and Sebag (Cornuéjols and Sebag, 2008), examined the experimental evidence relating to regular grammars, that is, to finite automata.

11.4 The covering test: random sampling in $\mathcal{H}$

In order to check the evidence for a phase transition, the standard procedure is as follows. One defines control parameters that correspond to key characteristics of the learning problem and then studies, in the space defined by these parameters, the probability that hypotheses of the type controlled by the parameter values can satisfy the constraints of training sets controlled in the same way. The same overall procedure was used in the study of first-order logic reported in Chapter 9.

The goal here is to test whether there exists a sharp transition in the coverage of training samples controlled by certain parameters (e.g., the length ℓ of the strings) when the characteristics of the hypotheses (e.g., the number of states, average branching factors, and so on) are varied.

The existence of such a sharp transition would be a strong indication that there exists some sort of discontinuity, in the "size" of the hypotheses (their coverage in the space of strings Σ^{ℓ}), that is intrinsic to the representation language associated with finite automata.

11.4.1 The experimental protocol

One key question is how to define a proper model for the random generation of automata and examples, in order to test the variations in the coverage of the automata (see the discussion on model RL in Chapter 9 for the case of relational learning).

Control parameters

Following the methodology introduced by Giordana and Saitta (2000), the phase transition phenomenon was investigated by means of *control parameters* chosen in accordance with the parameters used in the *Abbadingo* challenge (Lang *et al.*, 1998):[9]

- the number Q of states in the deterministic finite state automaton;
- the number B of output edges on each state;
- the number L of letters on each edge;
- the fraction a of accepting states, taken in $(0, 1)$;
- the size $|\Sigma|$ of alphabet considered;
- the length ℓ of the test examples and also the maximal length of the learning examples in $\mathcal{P}$ (as explained below).

The study first focused on the intrinsic properties of the search space without regard to the learning algorithms, that is, without regard to the lattice structure induced by state-merging operations. Using the set of control parameters, the average coverage of automata was studied analytically and empirically.

The experimental protocol

The sampling mechanism over the whole deterministic finite state automata (FSA) space was defined as follows. Given the control parameter values $(Q, B, L, a, |\Sigma|)$:

- for every state q, (i) B output edges (q, q') were created, where q' was uniformly selected with no replacement among the Q states; (ii) LB distinct letters were uniformly selected in Σ; and (iii) these letters were evenly distributed among the B edges above;
- every state q was turned with probability a into an accepting state.

The sampling mechanism for nondeterministic finite state automata differed from the above in a single respect: two edges with the same origin state were not required to carry distinct letters.

For each setting of the control parameters, 100 independent problem instances were constructed. For each FSA considered (the sampling mechanisms are detailed below), the coverage rate was measured as the percentage of covered examples in 1000 examples (strings of length ℓ) uniformly sampled.

[9] The Abbadingo challenge was proposed in 1997 in order to stimulate and evaluate research on the induction of target DFA from sets of training strings labeled by that target concept and a set of unlabeled testing strings. Each problem was to be considered solved by the first person to demonstrate a test-set error rate of 1% or less. The challenge comprised 16 benchmark problems of difficulty varying according to the size of the target concept (with 64, 128, 256, or 512 states) and the sparsity of the training data. The induction of target concepts with 128, 256, or 512 states remained unsolved for the case when the training data was lowest.

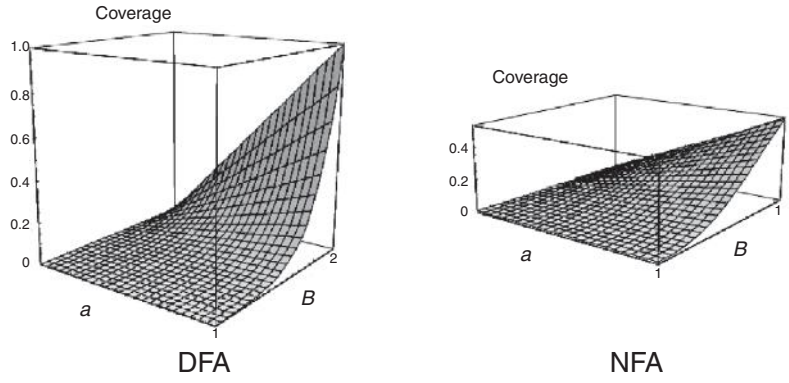


Figure 11.10 Coverage landscapes for deterministic and nondeterministic FSA, for $|\Sigma| = 2, L = 1$ and $\ell = 10$. The density of accepting states a varies in $[0, 1]$. The branching factor B varies in $\{1, 2\}$ for the DFA and in $\{1, 4\}$ for the NFA.

11.4.2 The findings

Figure 11.10 shows the average coverage in the (a, B) plane, for $|\Sigma| = 2$, $L = 1$ and $\ell = 10$, where the accepting rate a varies in $[0, 1]$ and the branching factor B varies in $\{1, 2\}$. Each point indicates the average coverage of a sample string s by an FSA (averaged over 100 FSA drawn with accepting rate a and branching factor B, tested on 1000 strings s of length ℓ).

These empirical results are stated analytically in the simple equations below, which give the probability that a string of length ℓ is accepted by an FSA defined on an alphabet of size $|\Sigma|$, with branching factor B and L letters on each edge, in the DFA and NFA cases (the number of states Q is irrelevant here).

$$P(\text{accept}) = \begin{cases} a(BL/|\Sigma|)^{\ell} & \text{for a DFA,} \\ a[1-(1-L/|\Sigma|)^{B}]^{\ell} & \text{for a NFA.} \end{cases} \tag{11.1}$$

The coverage of the FSA decreases as a and B decrease. The slope is more abrupt in the DFA case than in the NFA case; still, there is clearly no phase transition here.

No phase transition according to this protocol

While the reported results may seem too limited in their scope to warrant a definitive conclusion about the absence of a phase transition, in fact many more experiments with a wider range of parameter values all converge towards the same overall pattern of a gradually varying coverage probability. But the strongest argument comes from the analytical analysis and its near-perfect agreement with experimental measures. Clearly, there is no phase transition, with

respect to the covering test, when one looks at uniform sampling in the whole space of finite automata.

11.5 Learning, hypothesis sampling, and phase transitions

The coverage landscape displayed in Figure 11.10 might suggest that grammatical inference takes place in a well-behaved search space. However, grammatical inference algorithms do not explore the whole FSA space. Rather, as stated in Section 11.2, the search is restricted to the *generalization cone*, the set of generalizations of the prefix tree acceptor (PTA) formed from the set $\mathcal{P}$ of positive examples. The next step is thus to consider the search space actually explored by generic grammatical inference algorithms.

11.5.1 Evidence for abrupt changes when generalizing

A new experimental protocol

A new sampling mechanism was defined to explore the DFA generalization cone.

1. A number $|\mathcal{S}^+|$ (200 in the experiments) of examples of length ℓ were uniformly and independently sampled within the space of all strings of length ℓ (with varying ℓ), and the corresponding PTA was constructed.

2. A number N (50 in the experiments) of PTAs were constructed in this way.

3. A number K (20 in the experiments) of generalization paths, leading from each PTA to the most general FSA or the universal acceptor (UA), were constructed. In each generalization path ($A_0 = PTA(\mathcal{P}); A_1, \ldots, A_t = UA$), the ith FSA A_i was constructed from A_{i-1} by merging two uniformly selected states in A_{i-1} and subsequently applying the determinization operator if needed.

4. The generalization-cone sample for each training set $\mathcal{P}$ was taken from all the FSAs in all the generated generalization paths (about 270 000 FSAs in the experiments).

Regarding the NFA generalization problem, the sampling mechanism on the nondeterministic generalization cone differed from the above in a single respect: the determinization operator was never applied.

Figure 11.11 (left) shows the behaviour of the coverage in the DFA generalization cone for $|\Sigma| = 4$ and $\ell = 8$. Each DFA A is depicted as a point with coordinates (Q, c), where Q is the number of states of A and c is its coverage.

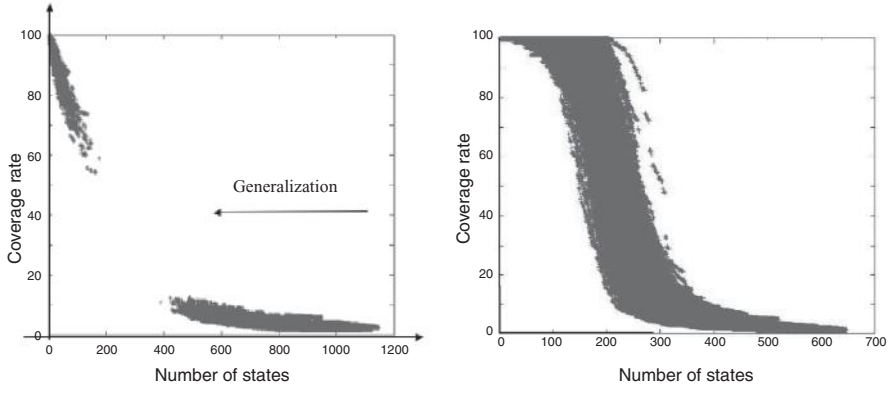


Figure 11.11 (Left) Coverage landscape *in the DFA generalization cone* ($|\Sigma| = 4$, $\ell = 8$, $|\mathcal{S}^+| = 200$). At the far right lie the 50 PTA sampled, with about 1150 states each. The generalization cone of each PTA includes 1000 generalization paths leading from the PTA to the universal acceptor. Each point indicates the coverage of a DFA, evaluated over a sample of 1000 strings. The graph shows the existence of a large gap regarding both the number of states and the coverage of the DFAs that can be reached by generalization. (Right) Coverage landscape *in the NFA generalization cone*, with same control parameters as in the left-hand panel.

The coverage rate for each FSA in the sample is evaluated from the coverage rate on 1000 test strings of length ℓ. Typical of all experimental results in the range of observation ($|\Sigma| = 2, 4, 8, 16$, and $\ell = 2, 4, 6, 8, 16, 17$), the figure shows a clear-cut phase transition. Specifically, here, the coverage abruptly jumps from circa 13% to 54%, and this jump coincides with a gap in the number of states of the DFAs in the generalization cone: no DFA with a number of states in the range $[180, 420]$ was found. The gap becomes even more dramatic as the length of the training and test sequences ℓ is increased.

A phase-transition-like phenomenon

Figure 11.11 (right) shows the behaviour of the coverage in the NFA generalisation cone, with $|\Sigma| = 4$ and $\ell = 16$. Interestingly, a much smoother picture appears. Although the coverage rapidly increases when the number of states decreases from 300 to 200, no gap can be seen either in the number of states or in the coverage rate itself.

Further experiments with different values for the control parameters confirm this general pattern (see Figure 11.12).

All the curves obtained in this new setting, where the effective search space is sampled, show a markedly different behavior from that in Section 11.4. In

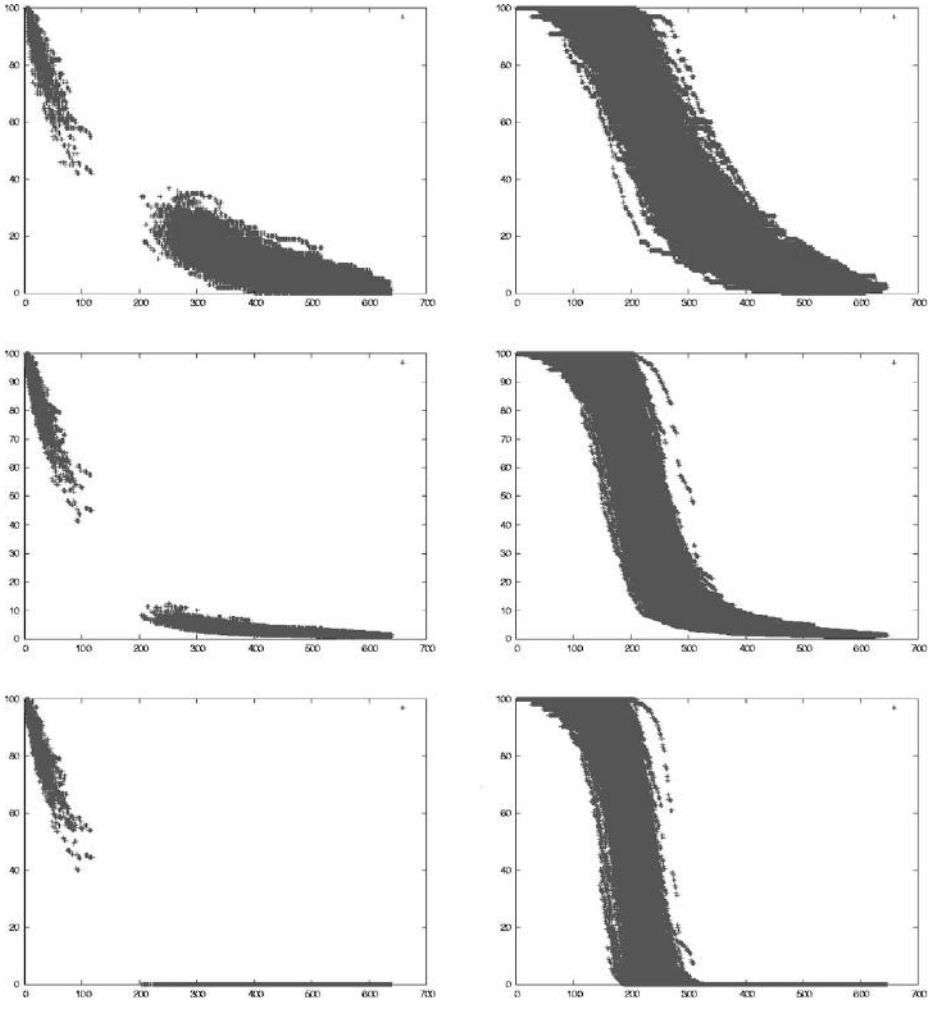


Figure 11.12 Here, coverage landscapes were obtained from 1000 experiments using an alphabet size $|\Sigma| = 4$, learning strings of length $\ell = 16$, and a training-set size $|S^+| = 100$. (Left) DFA; (right) NFA. The test strings are increasingly long from top to bottom with values 4, 16, and 32.

both the DFA and NFA cases, the covering rate varies rapidly at one point of the generalization process. This is all the more striking as the length of the strings in the training and test sets increases (see Figures 11.12 and 11.14).

Furthermore, even without more precise experiments, a large qualitative difference between the DFA and NFA cases manifests itself. It is easy to suspect its cause.

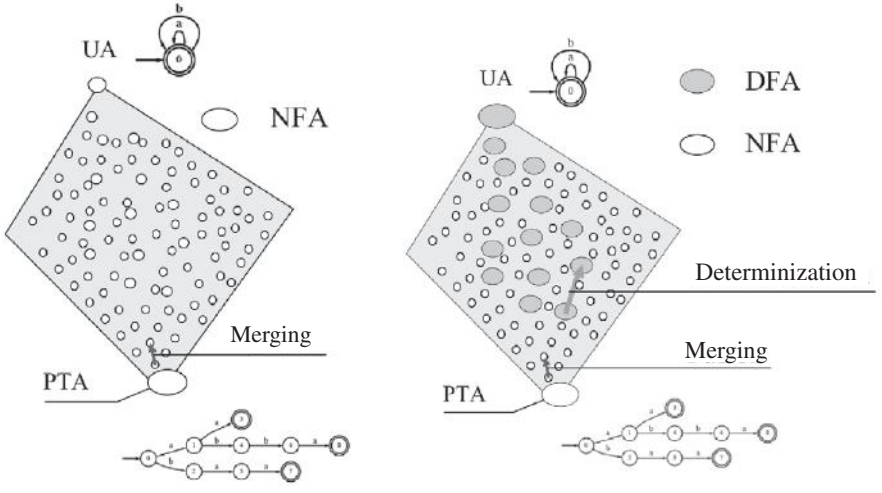


Figure 11.13 (Left) The lattice partition $Lat(PTA(S^+))$ is searched using merging operations. (Right) In the DFA case, further merging operations to restore determinization can cause jumps in the search space. At the top of each panel the universal algorithm (UA) is indicated.

Indeed, the generalization trajectories differ in the two cases in one respect, and this difference might be the key for the large observed differences. While, in both cases, learning uses state-merging operations starting from the PTA, in the DFA case further merging operations can occur at each step in order to restore the determinism of the candidate automaton (see Figure 11.13, which shows the difference between the two search spaces). We will examine later on whether this dissimilarity can explain the different learning landscapes. First, we will look more closely at the induction of NFA.

11.5.2 The generalization landscape in the NFA case

To get a more precise view of the evolution of the covering probability during the learning of a nondeterministic automaton (NFA), an extensive set of experiments were realized by Raymond Ros.[10] In these experiments he varied the following parameters:

Control parameters

[10]Raymond Ros was a Ph.D. student at L.R.I., University of Paris-Orsay, in 2005–6.

- the size of the alphabet $|\Sigma| \in \{2, 4, 8\}$;
- the length of the training and test strings $\ell \in \{[1, \ldots, 8], [1, \ldots, 16], 8, 16, 32\}$;
- the size of the training set $|\mathcal{P}| \in \{200, 500\}$;
- the size of the test set $\mathcal{T} \in \{200, 500, 1000\}$.

Again, the same pattern of an abrupt transition between hypotheses (automata) with low coverage and hypotheses with high coverage was observed (see Figure 11.14). These curves exhibit increasingly steeper transitions as the length of the learning and test strings increases. One analysis that predicts this is given in Appendix B. It is not known whether it is a correct explanation but it is an intriguing one. However, this tendency is a minor concern compared with the general pattern. Indeed, when one thinks about it, all these data are quite extraordinary and go exactly contrary to what might have been expected.

Seemingly a complete mystery

In fact, grammatical inference sets itself apart in the field of machine learning. As stated in Chapter 5, a major lesson of the theoretical study of learning over the last 30 years is that a learning bias is required if one wants to perform induction. Without a bias that limits the expressiveness of the hypothesis space, or more formally its capacity, one loosens the link between the empirical risk measured on the training set and the expected risk. Therefore, learning can no longer be guided and the hypotheses obtained are likely to perform almost randomly on the unseen examples. Because of this phenomenon, known as overfitting, there is a focus in machine learning on carefully controlling the so-called capacity or expressiveness of the hypothesis space, a problem also known as *model selection*. Margin-based learning methods, much in fashion nowadays, are a prominent example of new methods that seek to control capacity automatically.

Yet there exists no such representation bias in the case of the induction of regular languages. Regular languages are learned through exploration of the space of finite automata, and every regular language can be represented by at least one automaton. Furthermore, the exploration operator, by state-merging, is complete in the sense that, starting from the prefix tree acceptor $PTA(\mathcal{P})$, every generalization of the set of positive examples can be attained by a (well-chosen) succession of state merges. In formal terms, the capacity of the hypothesis language is infinite and the exploration operator *a priori* does not limit the effective search space. Therefore, if learning can succeed at all, its success must be explained on other grounds.

If one now turns to the characterization of the hypothesis space by the coverage rate of its elements, a further mystery is lurking around the corner.

. . . and even more so!

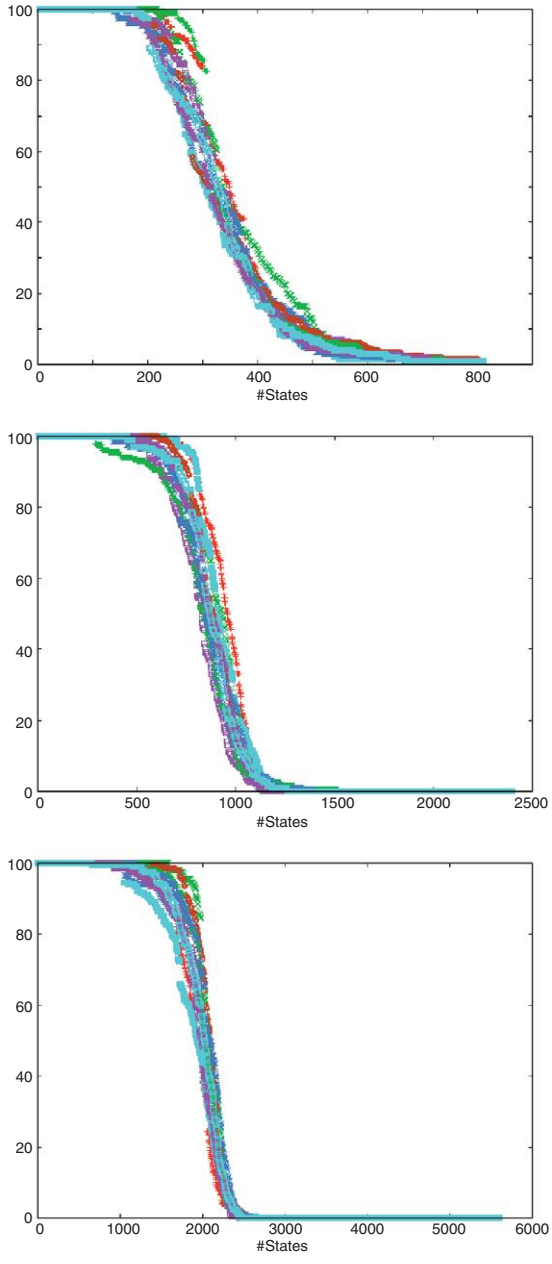


Figure 11.14 The coverage landscapes were obtained with the following control parameter values: $|\mathcal{P}| = 200$, alphabet size $|\Sigma| = 4$, and ℓ uniformly $\in [1, \ldots, 8], [1, \ldots, 16], [1, \ldots, 32]$, respectively, from top to bottom. The size of the prefix tree acceptor $PTA(\mathcal{P})$ correspondingly grows from approximately 800 states to approximately 2400 states and finally to approximately 5600 states. Each figure represents 10 learning trajectories, obtained starting with the same $PTA(\mathcal{P})$.

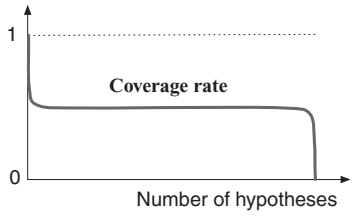


Figure 11.15 The coverage rate of the regular languages with respect to the strings of a given length ℓ.

Since the hypothesis space has no bias, every partition of the space of strings (i.e., $|\Sigma^\ell|$ for a given maximal length ℓ) can be represented. The number of partitions that cover n examples (strings) is equal to $\binom{|\Sigma^\ell|}{n}$. Therefore, the overwhelming majority of these partitions (languages) cover approximately $|\Sigma^\ell|/2$ examples. In consequence, a graph of the coverage rate of the elements of the language space should look like Figure 11.15, if one measures the coverage with respect to strings of length ℓ.

If one looks at the space of all generalizations of an automaton representing a prefix tree acceptor $PTA(\mathcal{P})$ (a generalization cone), the pattern is the same; almost all possible hypotheses cover approximately half the unseen examples.

Now, since the state-merging operator allows a full exploration of the generalization cone, one would expect that randomly sampled possible generalization trajectories would predominantly explore languages with a coverage rate close to $1/2$. The experiments tell a completely different story. The learning trajectories first stay for a long time in the region of languages that cover a small fraction of all possible strings and then suddenly jump upward, rapidly crossing the region of intermediate coverage, before leveling off in the region of languages that accept almost all strings.

This surprising behavior points to special properties of the merging operations. At one point in the exploration, a few state merges suffice to change radically the coverage rate of the generated automata.

One must realize that, even though this closely resembles the behavior described in Chapters 9 and 10, the reason behind it is quite different.

Towards solving the mystery

In the case of inductive logic programming (ILP) the phase transition behavior is *intrinsic to the representation language*. When sampled uniformly with respect to the four control parameters, there is a very narrow, vanishingly small, region with hypotheses of intermediate coverage. This has nothing to do with the learning algorithms. By contrast, in grammatical inference, because there is no *a priori* representation bias, most potential hypotheses cover approximately half all possible strings. However, it is the generalization operator which induces an

abrupt change in the coverage rate of the hypotheses generated during learning and a fast crossing of the region where most potential hypotheses lie.

In grammatical inference, *the phase transition is a property of the learning approach*, not of the space of hypotheses. This fact is even more striking in the case of DFA learning.

11.5.3 The generalization landscape in the DFA case

In the NFA case we have seen that, the generalization cone starting from a learning sample is explored by successive generalization steps, i.e., state merging. The coverage rate always undergoes a sharp transition between a regime where the induced automata cover a very small fraction of random test strings and a regime where they cover a large fraction of them. The first experiments reported in Section 11.5.1 clearly indicate that even though an abrupt transition occurs also in the case of the inference of DFA, it is of a different nature. Specifically, a gap appears between the automata with low coverage rate that are produced first and automata of high coverage rate that are encountered later in the generalization process.

It is of interest to look again at the *Abbadingo* challenge presented in Section 11.4.1. One challenge regarding the induction of DFA involved a target automata of 128 states with a "sparse" learning set of $|\mathcal{P}| = 4382$ strings defined over an alphabet of size $|\Sigma| = 2$. The strings had an average length $\ell = 17$. This problem remained unsolved during the competition. A look at the variations in the coverage rate in the generalization cone may explain part of the reason for this (see Figure 11.16). As is strikingly apparent, no DFA of coverage rate between almost 0% and 95% is ever considered. And this curve is one among hundreds that exhibit the same pattern. If, indeed, this curve represents the behavior of the learning algorithms, then it is no wonder that induction fails in this case. A set of experiments was therefore aimed at a better characterization of this phenomenon in the DFA case.

It is noticeable that the same shape was found for the variation in the coverage rate in a large variety of situations (see Figures 11.12, 11.16, 11.17–11.19). It is therefore of a generic nature, i.e., independent of the size $|\Sigma|$ of the alphabet, the length of the learning and test strings, or the size of the training set.

Since this shape exhibits a large gap in the coverage of the automata that are considered in the generalization cone, serious consequences can be expected for the learning algorithms. It is therefore important to understand the reasons for this typical behavior.

Why is there a gap?

We start by examining some parameters associated with the dynamics of the generalization process. An important characteristic associated with graphs and

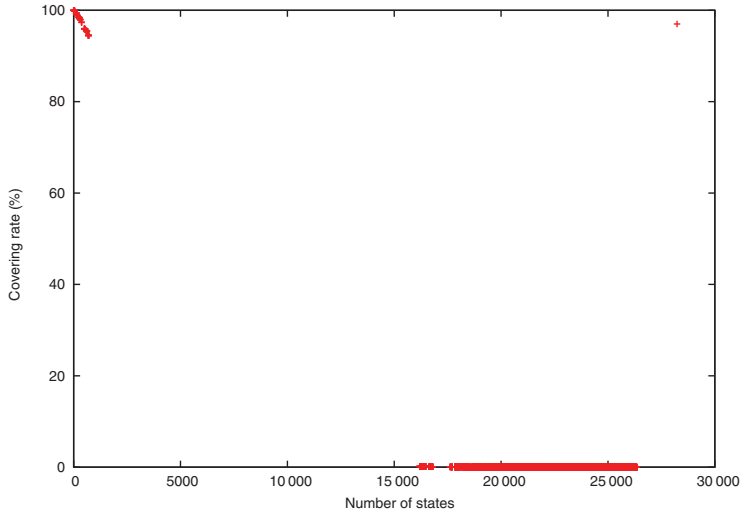


Figure 11.16 Induction of a DFA in the conditions of the *Abbadingo* challenge. The alphabet size $|\Sigma| = 2$, the average length of the strings $\ell = 17$, and the target automaton has 128 states and is used to label the 4382 learning strings. The prefix tree acceptor in this case comprised approximately $27\,000$ states.

with finite state automata in particular is *the ratio of the number of exit edges and the number of (non-terminal) states*. We would expect it to increase as states are merged since at each time the resulting merged state inherits at least as many edges as the more connected parent state. Indeed, this is what is observed experimentally. For instance, Figure 11.18, obtained for an alphabet size $|\Sigma| = 4$, a training set size $|\mathcal{P}| = 500$, and strings of length $\ell \in (1, \ldots, 8)$, shows a gradual increase in the ratio $\#edges/\#states$. In the PTA the ratio is close to 1 since the automaton essentially consists of sequences of states associated with each training string. This is especially true when the training set is sparse in the space of all possible training strings. Then, for instance, the space of strings of length $\ell \in (1, \ldots, 8)$ defined over an alphabet of size 4 is of size

$$\sum_{i=1}^{8} 4^i = \frac{4(4^8 - 1)}{4 - 1} = 87\,380.$$

A learning set of size 500 is therefore sparse.

The surprising fact is that there is no great jump in the ratio when the gap in the learning trajectories is crossed. In Figure 11.18, it goes from a value ≈ 1.6 to a value ≈ 2. Only then does it sharply increase, during the last generalization operations.

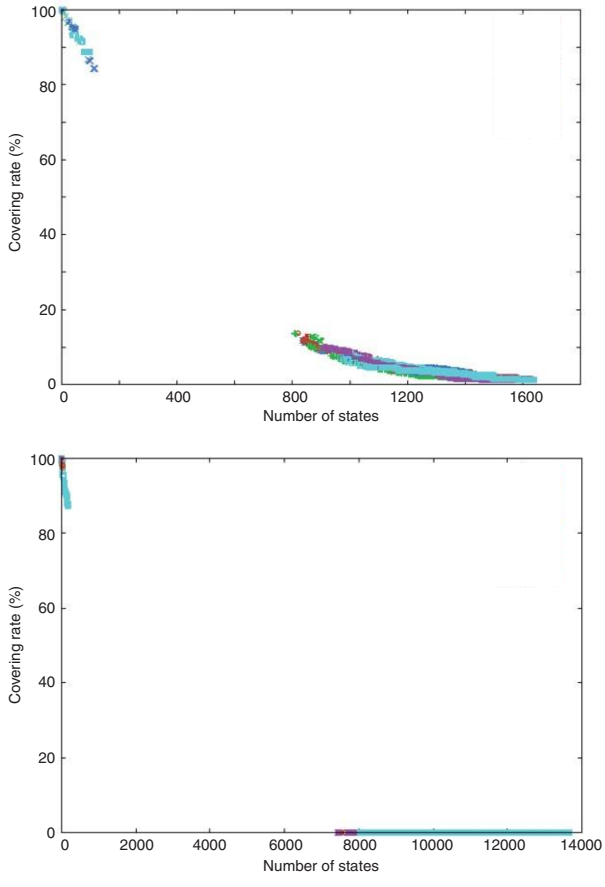


Figure 11.17 Variation in the coverage rate when random samples of automata are taken within the generalization cone from the DFA. Here the alphabet size $|\Sigma| = 4$ and the learning set size $|\mathcal{P}| = 500$. In both figures the profile is made up of eight generalization processes, in the top panel with strings of length $\ell \in (1, \ldots, 8)$ and in the bottom panel with strings of length $\ell \in (1, \ldots, 32)$.

Another key parameter is *the ratio of the number of accepting states and the total number of states*. Since each time an accepting state is merged with another state the resulting state is accepting as well, one would expect this ratio to increase during the generalization process. However, experiments tell a completely different story (see Figure 11.19). Whereas the ratio does indeed gradually increase during the first part of the generalization, before the gap, it jumps back to almost 0 after the gap. How should we to understand this?

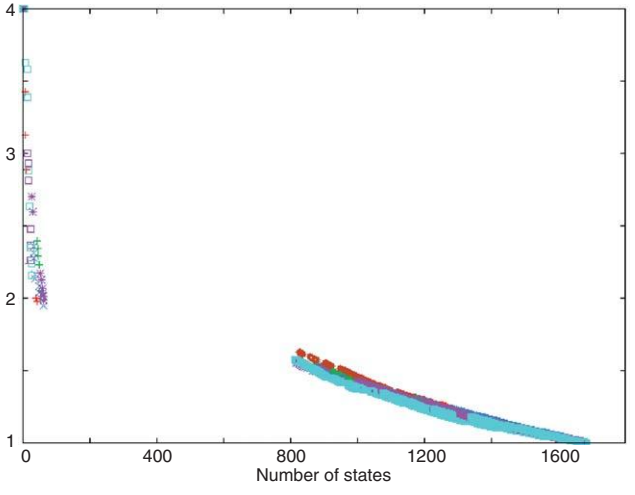


Figure 11.18 Evolution of the ratio of the number of exit edges and the number of (non-terminal) states. Here, the alphabet size is $|\Sigma| = 4$, the training set size is $|\mathcal{P}| = 500$ and the strings have length $\ell \in (1, \ldots, 8)$. The curve results from the aggregation of eight generalization trajectories.

One explanation would be that a small number of accepting states is involved in all the state-merging operations. In order to test this hypothesis, a lengthy visual study of the evolution of the automata was undertaken. An example of an automaton after the gap is given in Figure 11.20. The states have an array proportional to their connectivity. One can see that one terminal and accepting state has attracted most links. Therefore, the ratio of accepting states and the total number of states remains low and can even decrease dramatically.

One fact that differentiates DFA induction from NFA induction is that, in the former, after each state-merging operation other such operations may be necessary to restore the deterministic character of the automaton. This is illustrated in Figure 11.21. It is then natural to examine how many of these determinization operations are carried at each step in a generalization trajectory.

In Figure 11.22, the y-axis reports the number of states merging for determinization after each generalization step starting, as usual, from the PTA. Again, this figure is a compound profile with eight individual learning trajectories. The pattern is always the same. Up to the threshold of the gap, the number of determinization operations is very low, limited to less than 10 except in two cases, one where an "eruption" of approximately 100 operations occurs on one trajectory and another where approximately 160 operations occur on another trajectory. However, at the verge of the gap, for all trajectories there occurs a sudden

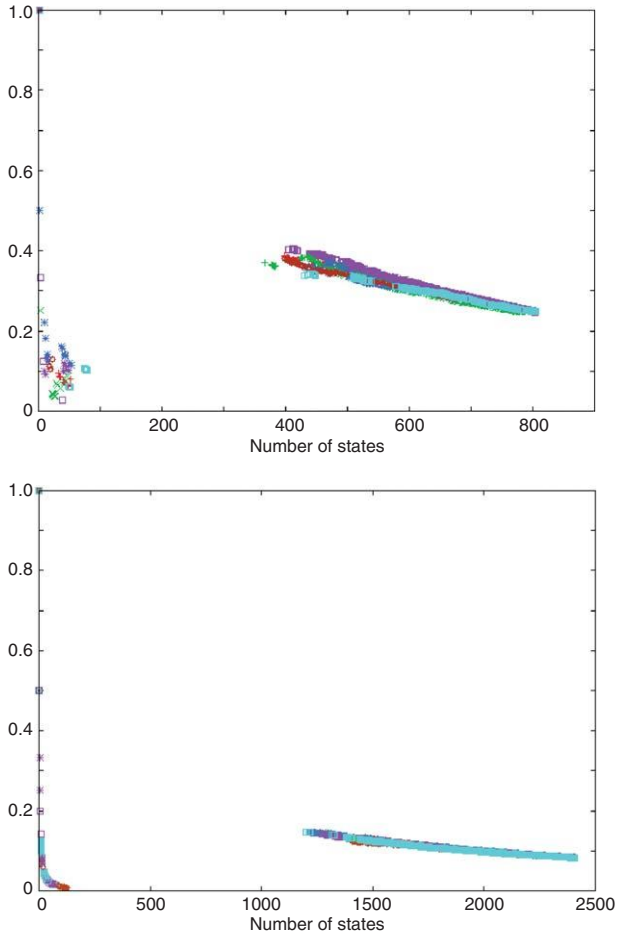


Figure 11.19 Evolution of the ratio of the number of accepting states and the total number of states. Here, the alphabet size $|\Sigma| = 4$ and the training set size $|\mathcal{P}| = 200$. The curves result from the agglomeration of eight curves. In the upper panel the strings are of length $\ell \in (1, \ldots, 8)$. In the lower panel the strings are of length $\ell \in (1, \ldots, 16)$.

very large cascade of determinization operations of sizes, in these cases, between 1200 and 1500 operations. This is what explains the gap in the exploration of DFA by generalization from the PTA. After the gap, again each generalization operation entails very few determinization operations.

Attempt at a theoretical model

Given the apparently universal character of this singular pattern, it is important to look at models that could predict it. At the elementary level, a

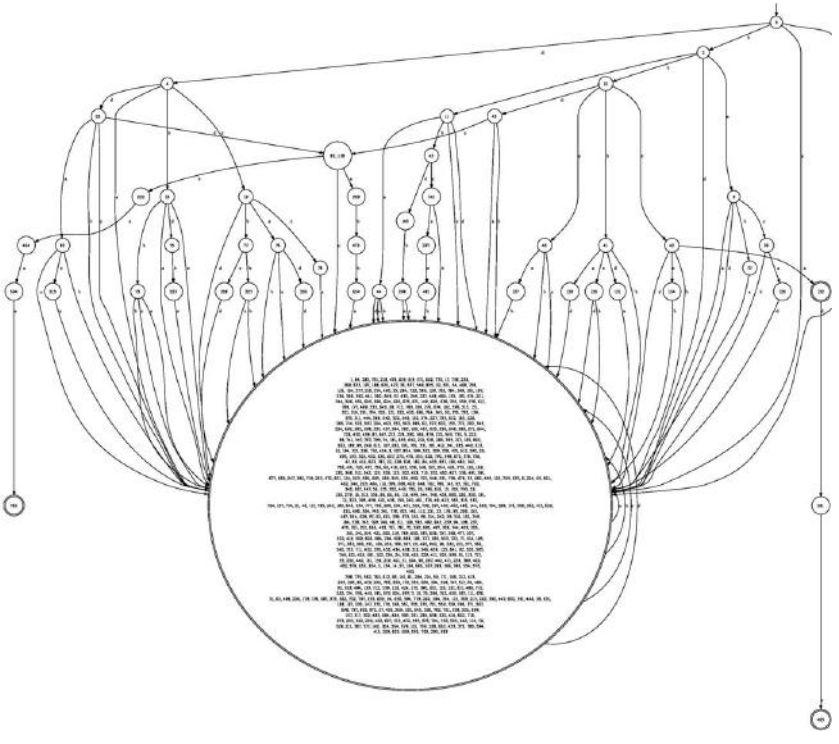


Figure 11.20 An example of an automaton just after the gap in the generalization trajectory has been crossed. One terminal state has attracted most of the links.

determinization merging takes place whenever two states are merged that have at least one output link with the same letter (see Figure 11.21, where the merged states share "*a*" as an output letter). Indeed, in this case the output of the state is no longer determined if a letter is given. This, in turn, entails other state-merging operations further down on the states reached by the same letter. The expected number of state-merging operations $\mathbb{E}[\text{merges}]$ can be computed from a chain-reaction-like model:

$$\mathbb{E}[\text{merges}] \;=\; 1 \;+\; \mathbb{E}[\text{merges}] \left(\sum_{i=1}^{|\Sigma|} \mathtt{P}(k=i) \cdot i \right) \tag{11.2}$$

where $\mathtt{P}(k = l)$ is the probability that a state-merging operation provokes an intersection of size i of the output letters for the two merged states. The expected number $\mathbb{E}[\text{merges}]$ diverges when $\sum_{i=1}^{|\Sigma|} \mathtt{P}(k=i) \cdot i > 1$.

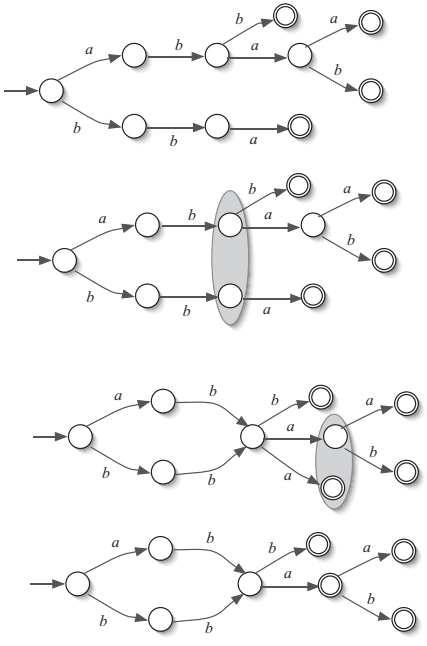


Figure 11.21 Starting from the learning sample $\mathcal{S} = \{\langle a, b, c\rangle, \langle a, b, a, a\rangle, \langle a, b, a, b\rangle, \langle b, b, a\rangle\}$, the prefix tree acceptor (top automaton) is built. Then, a pair of states is chosen for merging (shaded oval). This leads to the third automaton from the top. This automaton being non-deterministic, another state-merging operation is carried out to restore the deterministic character of the automaton (bottom).

This model is crude in the sense that it assumes that the state-merging operations are independent. As it is, it does not take into account the distribution of the letters and of the different states with respect to their output connectivity. However, it is easy to turn this simple model into a simulation tool where one starts by providing the relative proportion of states with $0, 1, 2, \ldots$ output links and then running the simulation.

That was done, for instance, for the case of the *Abbadingo* challenge already encountered in Figure 11.16. There, the alphabet size is $|\Sigma| = 2$, the average length of the strings is $\ell = 17$, the target automaton has 128 states and is used to label the 4382 learning strings. The prefix tree acceptor in this case comprised approximately $27\,000$ states. The onset of the gap occurs when the considered automaton has approximately $16\,200$ states, that is, after approximately 8800 state-merging operations. This is to be compared with the prediction of the model, which is that a divergence in the expected number of state-merging operations should arise after $11\,716$ state-merging operations.

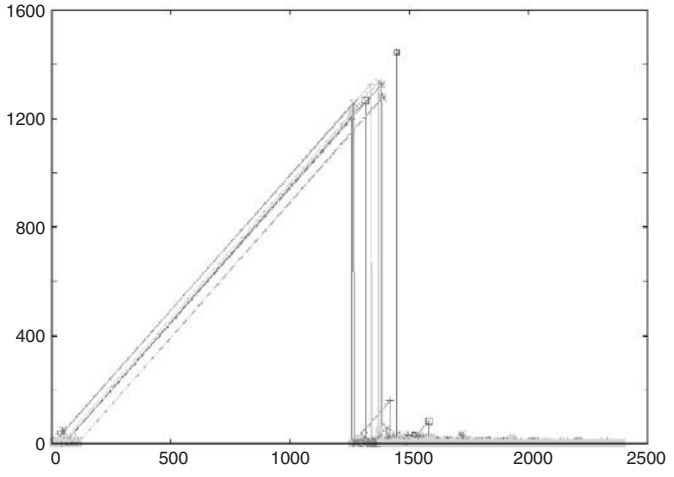


Figure 11.22 The number of states of the DFA that are produced during an exploration by generalization from the PTA (going from right to left) is plotted along the x-axis. The number of state-merging operations required to restore the deterministic character of the new candidate automaton after each generalization step is plotted along the y-axis. One large cascade of state-merging operations occurs at the verge of the gap.

A quite remarkably precise prediction

The prediction is in error by only 33%. Given the simplifying assumptions that underly the model, the agreement with the actual value is quite remarkable.

The existence of a gap in the coverage rate of the candidate DFA generated by a generalization process from the PTA can thus be explained in terms of a chain-reaction-like phenomenon. At one point, one more state-merging operation leads to a state of the system, i.e., the current candidate automaton, that cannot be made deterministic without a very large number of determinization operations. This effectively renders the generalization process blind to a whole region of the generalization space, since it cannot stop generalizing in this region. In other words, there is a long sequence of generalization steps, each producing an NFA, before a DFA is obtained. This, in effect, means that learning methods based on state-merging operations are unable to produce a DFA in a whole region of the generalization space.

In the next section we study the consequences of the performances of these learning algorithms. Are these generalization performances really as bad as might be expected from the profile of the coverage rate during generalization?

In the following, we focus on the induction of DFAs.

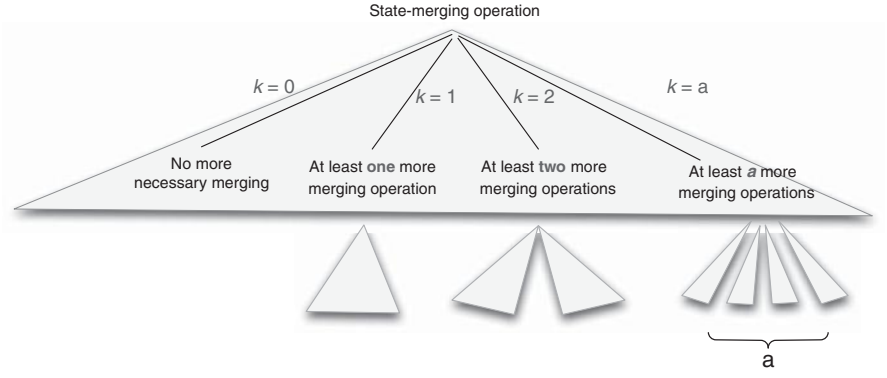


Figure 11.23 When two states are merged they can have $0, 1, 2$ or more output letters in common ($k = i$). Depending on the size k of this intersection, $0, 1, 2$ or more further determinization mergings must take place. This can lead to a diverging chain reaction of state-merging operations, each new operation potentially leading to more.

11.6 Consequences of the behavior of the learning algorithms: how bad is it?

The coverage landscape, for the DFAs, shows a hole in the generalization cone: for a large coverage range, the density of hypotheses falls abruptly. Therefore, a random exploration of the generalization cone would face severe difficulties in finding a hypothesis in this region and would be likely to return hypotheses of poor performance if the target grammar had a coverage rate in this "no man's land" interval. However, the existing learning algorithms do not merge states at random when generalizing. Do their control strategies protect them against the avalanche phenomenon and allow them somehow to explore the "gap"?

Does the phase-transition-like phenomenon actually hamper learning?

Another concern is related to what has been observed in inductive logic programming, namely a discrepancy between the target concept and the learned hypotheses. In the case of the induction of finite state automata, a first and easy measure of such discrepancies is the difference in the coverage rate of the target automaton and the learned one. For instance, given a target automaton with coverage lying in the gap (*e.g.*, 50%), are the learning algorithms able to probe the associated region of hypotheses having same range of coverage rate and to return a hypothesis in this region? In other words, are they able to guide the search toward hypotheses of appropriate coverage rate, especially if this coverage falls in the gap?

The performances of two standard algorithms in grammatical inference, namely the RPNI and the RED–BLUE (on EDSM) algorithms (Oncina and García, 1992; Lang *et al.*, 1998) have thus been studied. We first report the experiments and their outcome. We then look at the control strategies used by these algorithms.

11.6.1 Experimental setting

The experiments reported in Section 11.5 were for training sets made of positive-only string sequences. Since the focus was on the generalization cone based on the PTA, there was no need to consider negative instances.

However, in order to assess the performance of a learning algorithm, the learned hypothesis must be compared with the target automaton. Therefore, another experimental setting is used in this section: the sampling of target automata and the construction of training and test sets. These data sets include positive and negative examples, as most grammatical inference algorithms (and specifically RPNI and RED–BLUE) use negative examples as a means of stopping the generalization process.

The first experiments tested whether heuristically guided inference algorithms can find good approximations of target automata when these automata have a coverage rate falling in the gap and when they have the low coverage rate typical of many applications. Thus, target automata with (i) an approximately 50% coverage rate (as in the influential *Abbadingo* challenge and in the middle of the "gap") and (ii) with a 3% coverage rate were considered.

For each target coverage rate, the same experimental setting as that described in Lang *et al.* (1998) was used in order to retain a certain number of target automata with a mean size of Q states ($Q = 50$, in these experiments). Then, for each automaton, $N = 20$ training sets of size $|\mathcal{S}_L| = 100$, labeled according to the target automaton, were generated, with an equal number of positive and negative instances ($|\mathcal{P}| = |\mathcal{N}| = 50$) of length $\ell = 14$. The coverage rate was computed as before on 1000 uniformly chosen strings having no intersection with the training set.

11.6.2 The coverage rates of the target and learned automata

Let us first compare the coverage rate of the learned automata with the coverage rate of the target automata.

In the *first set of experiments*, target automata were generated having a coverage rate close to 56%. Only the graph obtained for the RPNI algorithm is shown here (see Figure 11.24), with three typical learning trajectories. Similar results were obtained with the RED–BLUE algorithm.

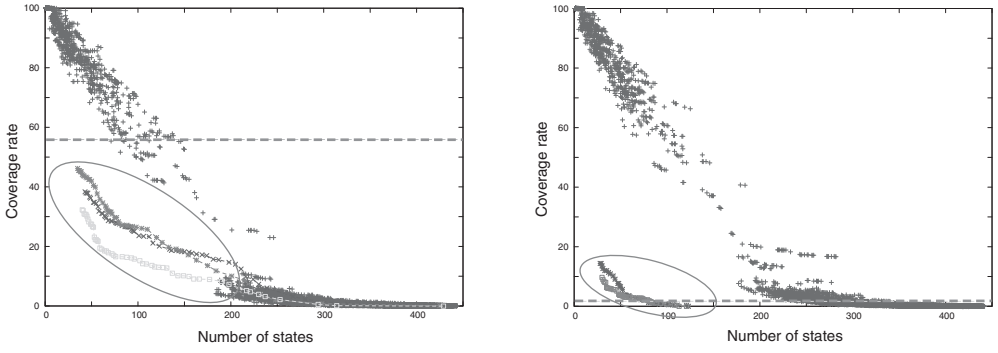


Figure 11.24 (Left) Three RPNI learning trajectories for a target automaton of coverage 56%. Their extremity is outlined in the light gray oval on the left. The broken horizontal line corresponds to the target automaton coverage. The cloud of points corresponds to random trajectories. (Right) Same as the left-hand panel except for the coverage of the target automaton, here 3%.

One immediate finding is that both the RPNI and the EDSM heuristics manage to probe the "gap". The three learning trajectories obtained using the RPNI algorithm show that candidate automata of all coverage rates are generated up to approximately 40% coverage rate. This contrasts with the learning trajectories produced from the same prefix tree acceptors when the state-merging operations are random; here again, a wide gap appears.

This may explain why the gap phenomenon was not discovered before, and why the RED–BLUE algorithm, for instance, could solve some cases of the *Abbadingo* challenge where the target automata have a coverage rate of approximately 50%. The RPNI algorithm, however, tends to overspecialize the target automaton by returning learned automata with coverage in the range (35%, 45%) when the target automata have 56% coverage rate. Conversely, experiments show that RED–BLUE tends to overgeneralize by 5% to 10%.

A *second set of experiments* was carried out in which the target automata had a coverage rate of approximately 3%. The results (Figure 11.24) show that, in this case, RPNI ends up with automata of coverage 4–6 times greater than the target coverage. The effect is even more pronounced with RED–BLUE, which returns automata having an average coverage rate around 30%!

These results are unexpected, since low-coverage automata seem to be much more densely probed than medium-coverage ones. An explanation is needed. But before looking for it, we will examine further the generalization performances of learned automata in this low-coverage regime.

11.6.3 Generalization error

A set of experiments analyzed the learning performances of the algorithms with respect to test errors, differentiating the false positives and the false negatives.

In these experiments, the type of target automaton was chosen by setting the number of states Q and some predetermined structural properties, as follows.

Deterministic target automata were generated along four parameters:

- the size $|\Sigma|$ of the alphabet;
- the number of states $|Q|$;
- the *density* of the connections;
- the level *Trec* of *recursivity* of the connections.

More precisely, for each generated target automaton a tree of $|Q|$ states was first built randomly. At this stage, the density of connection is minimal, with value 1. Then *density* $\times|Q|(|\Sigma|-1)$ transitions are added, of which a number *Trec* create cycles. The leaves of the tree are taken as accepting states, and the remaining states are labeled as accepting with a probability of 20%.

A whole set of experiments was carried out by varying the four control parameters. We report here one typical finding where the number of states of the target automaton is the key parameter. In these experiments, $|\Sigma| = 4$, *Trec* $= 50\%$, *density* $= 50\%$. Moreover, the learning set consisted of 1000 strings of length $\ell \in (1, (2 \times depth))$, where $depth$ is the depth of the target automaton (the maximal depth of the initial tree).

It is useful to measure the degree of structural completeness achieved by the learning set, that is, the proportion of all transitions of the target automaton that are excited by the learning strings. The following results were obtained for training sets of structural completeness *Prct* $\geq 40\%$. The *uniform coverage rates* $ucov_t$ and $ucov_f$ were estimated using 1000 test strings of length $\ell \in (1, (2 \times depth))$ that were not used in the training set. The *positive coverage rate* $pcov_f$ (the *negative coverage rate* $ncov_f$) is estimated using a sample of 1000 test strings labeled as positive (as negative). The numbers in Table 11.1 were obtained by repeating each experiment 100 times for each setting of the control parameters.

Table 11.1, obtained for different sizes of the target automaton and for training sets of structural completeness above 40%, confirms that both RPNI and RED–BLUE return overgeneralized automata.

Indeed, what is remarkable is that this overgeneralization *does not imply* that the learned automata are complete: on the contrary, the coverage of the positive examples remains below 65% in all but one setting. On the one hand their average coverage is vastly greater than the coverage of the target automaton; on the

Table 11.1 Performances of the algorithms RED–BLUE (RB) and RPNI for target DFA of sizes $Q = 15, 25, 50$, and 100 states and of (low) coverage rate $ucov_t$. The learned automata are characterized by Q_f, $ucov_f$, $pcov_f$, and $ncov_f$, which respectively denote their average size, their average coverage, the true positive rate, and the false positive rate.

Algorithm	Q	$ucov_t$	Q_f	$ucov_f$	$pcov_f$	$ncov_f$
RB	15	5.97	10.38	33.81	60.93	34.69
RB	25	4.88	12.77	40.35	62.68	37.87
RB	50	4.20	14.23	45.38	66.14	42.23
RB	100	3.39	13.13	30.35	42.81	28.69
RPNI	15	5.95	5.14	22.90	57.51	26.99
RPNI	25	4.70	7.56	23.07	56.38	25.98
RPNI	50	3.87	14.08	23.45	51.89	24.42
RPNI	100	3.12	26.41	23.15	50.12	24.40

other hand they tend to cover only part of the positive test instances while they cover a significant proportion of the negative test instances. Both *precision* and *recall* are therefore poor. This shows that the heuristics used in both RPNI and RED–BLUE may be inadequate for target automata of low coverage. It is time to examine these heuristics and why they can lead to such results.

A poor performance in the case of targets with low coverage

11.6.4 The control strategies and their impact

The RPNI (*regular positive and negative grammatical inference*) algorithm, (Oncina and García, 1992), starts by building the prefix tree acceptor (PTA) from the positive instances of the training set. The algorithm then iteratively selects pairs of states that could be merged, checks whether a candidate-merge would yield an automaton that covers at least one negative instance, and makes the merge if it is admissible. The choice of a pair of states to merge is made systematically and does not depend on the current situation of the search. When the PTA is built, its states are given labels $\{q_1, q_2, \ldots, \}$ starting from the root and increasing in a breath-first manner. The RPNI algorithm considers pairs of candidate states according to their lexicographic order, that is, it favors merges that are close to the root of the current automaton.

If one looks at the size of the generalization steps that this heuristic entails, one can see that merges of pairs of states close to the root lead to larger generalization steps than merges of pairs of far-away states. Furthermore, RPNI is subject to an "avalanche" of determinization operations.

The heuristic developed for the Red–Blue algorithm, called EDSM (*evidence driven state merging*), by contrast chooses at each step the pair of states that will entail the largest number of determinization operations. The mean generalization step is therefore generally higher than for RPNI. However, it is less prone to the avalanche phenomenon. This last quality may also explain its tendency to overgeneralization.

Indeed, in the case of the RPNI algorithm, during the last steps of the generalization process most attempted merges are pruned because they would lead to an avalanche of further merging operations and therefore to the covering of negative instances. Consequently, the generalization process stops short of too much overgeneralization. This is confirmed by the experimental results. The Red–Blue algorithm, however, can apply generalization steps further away before the generalization process stops. At least in the case of target automata having small coverage rates, this unfortunately steers the learning towards hypotheses of much larger coverage, that is, to high rates of false positive predictions.

11.7 Comments

We now compare and contrast the cases of learning in first-order logic and in grammatical inference. In the former the phase-transition-like phenomenon is inherent in the language representation: when hypotheses and examples are drawn randomly according to a uniform distribution within the space of the control parameters, an abrupt transition is observed between a region where the hypotheses cover almost all examples and a region where almost no examples are covered.

When the same kind of experiment is carried out in grammatical inference, looking at the probability that random automata recognize random strings, no such sudden transition is observed. The average branching factor of the states, i.e., the average number of output links, plays a larger role in the probability of coverage but, overall, the coverage rate exhibits a well-behaved gradient that should perfectly inform the learning algorithms.

However, when the *modus operandi* of the most prevalent learning systems are taken into account, the findings are quite different. When the exploration of the hypothesis space starts from the prefix tree acceptor and proceeds by the merging of states in the candidate automaton then, again, a very steep transition appears between the beginning of the search where candidate automata cover almost no test strings and the region reached later where almost every test string is covered. Actually, the phenomenon is even more dramatic with the induction of DFAs since then an actual gap usually appears between the two regions.

In grammatical inference, therefore, the phase transition phenomenon is not inherent in the language representation: rather, it is due to the generalization process used in learning. Somehow the state-merging operations used for generalization radically increase the gradient in the coverage of the automata that are considered. In addition, as was seen in the case of DFA induction, a further phenomenon occurs: an avalanche of state-merging operations needed to ensure that candidate automata are deterministic.

A phase transition due to the learning algorithm, not the representation language.

The nature of the phase transition in grammatical inference is therefore different from that encountered in first-order-logic learning. It is a property of the learning algorithm rather than a property of the representation language.

Moreover, whereas the covering test is **NP**–complete in first order logic, it is of linear complexity in the case of grammatical inference. Consequently, one does not observe an increase in the search cost around the phase transition.

Grammatical inference does not truly belong to the same class of problems as SAT or relational-concept learning and thus does not have the same kind of phase transition. But, in a strange way, the phase transition exhibited in DFA is a nice example of a physical-like process with a chain reaction mechanism. In some respects, then, it is a proper example of a phase transition.

12

Phase transitions in complex systems

Contents

Even though this book is focused on learning, we thought it might be useful to widen its scope to include a brief overview of the emergence of ensemble phenomena (typically phase transitions) in complex networks. Beside their intrinsic interest as systems amenable to be studied via statistical physics methods, complex networks may well impact relational learning because both examples and formulas can be represented as networks of tuples, as we have seen in previous chapters. The same can be said for the constraint graph in CSP and the factor graph in SAT. In ensemble phenomena, emerging from a network of "microscopic" interactions, it is likely that the underlying graph structure has an impact on the observed macroscopic properties; thus cross-fertilization might be of mutual benefit.

Complex networks are *complex systems*, meaning systems composed of a large number of mutually interacting components. Even though in such systems it is impossible to describe the behavior of the individual components, the patterns of interactions among them allows macroscopic properties to emerge which would be missed by a reductionist approach. Thus, the science of complexity

aims to discover the nature of these emerging behaviors and to link them to the system's microscopic level description.

12.1 Complex systems

From its very definition, it is clear that the science of complexity naturally derives from statistical physics. Thus, the emergence of phase transitions and the behavior of complex systems have strong links. In fact, both derive their properties from the effects of the interaction patterns among many small components, and both have at their core the notion of graphs or networks. The nodes of such a network constitute an ensemble of elements, whose interactions let the network's macroscopic behavior emerge.

In Chapter 3 we introduced the notion of graphs and mentioned the best known classes investigated in mathematics and the most common structures occurring in the real world. As mentioned in that chapter, a first hint as to the existence of phase transitions in graphs was provided by Erdös and Renyi (1959), who discovered what amounted to a phase transition in the connectivity during the process of generating a random graph from the $\mathcal{G}_{n,p}$ ensemble.

Later, the emergence of phase transitions in complex networks was investigated systematically. An overview of early results was given by Newman (2003). Several authors have studied spin models, such as the Ising model, on networks of various kinds, for example random graphs (Dorogovtsev *et al.*, 2002; Leone *et al.*, 2002), small-world networks (Barrat and Weigt, 2000; Pelalski, 2000; Herrero, 2002), and Barabàsi–Albert networks (Aleksiejuk *et al.*, 2002).

The question behind this body of work is whether (and how) graph structures that are not random affect the emergence and location of the phase transition. For instance, Walsh (1999) looked at colorability in Watts–Strogatz small-world graphs; he found that these graphs are easily colorable for both small and large values of the shortcut density parameter p but that they are harder to color in intermediate regimes. Vàzquez and Weigt (2003) examined the vertex cover problem, and found that on generalized random graphs solutions are harder to find for networks with stronger degree correlations.

Given the links between complex networks and statistical physics, there is no wonder that methods borrowed from the latter are used to investigate properties of the former. For instance, the continuum theories proposed to predict the degree distribution can be mapped, often exactly, onto some well-known problems investigated in statistical physics. Two such cases are the mapping between evolving networks and Simon's model (Simon, 1955), on the one hand, and the equilibrium Bose–Einstein gas (Park and Newman, 2004), on the other.

Park and Newman (2004) proposed a method for generating networks that match the expected properties of a graph ensemble, given a set of measurements made on a real-world network. As for the Boltzmann distribution in statistical mechanics, these models offer the best prediction of properties subject to the constraints imposed by a given set of observations. Using exponential random graphs, Park and Newman provided for models in this class exact solutions that show arbitrary degree distributions and arbitrary but independent edge probabilities. They also discussed more complex examples with correlated edges; for these cases approximate or exact solutions can be found by adapting various methods from statistical physics.

An interesting approach was proposed by Biely and Thurner (2006). They used a microscopic Hamiltonian derived from the socio-economically motivated assumption that individual nodes increase or decrease their utility by linking to nodes with respectively a higher or lower degree than their own. Utility is an equivalent of energy in physical systems. Nodes tend to maximize utility as physical particles minimize energy. From the study of the temperature dependence of the emerging networks, the existence of a critical temperature T_{cr} was observed, at which total energy (utility) and network architecture undergo radical changes. At this topological transition a scale-free network, with complex hierarchical topology, is obtained.

Another approach explicitly based on statistical physics was proposed by Vicsek (2007). He investigated topological transitions in the restructuring of networks. In his approach energy is associated with the different network topologies and temperature is used as a measure of the noise level during rewiring of the edges. A variety of topological phase transitions emerges when the temperature is varied. These transitions denote changes in the essential features of the global structure. Then Vicsek addressed the question of network structure modularity. The global organization can be viewed as the coexistence of structural *communities*, associated with highly interconnected parts. The approach he proposed allows one to analyze the main statistical features of the interwoven sets of overlapping communities, so making a step towards the uncovering of the modular structure of complex systems. The approach is based on defining communities as clusters of percolating complete subgraphs called k-cliques. A set of new characteristic quantities is defined for the statistics of the communities, and an efficient technique is applied to explore overlapping communities on a large scale. Significant overlappings among communities were detected, and some universal features of networks uncovered.

Order and disorder

A key concept in complex systems, as well as in statistical physics, is that of *order*. Several types of phase transition, occurring in a variety of fields, concern a transition between ordered and disordered states. Different statistical mechanics tools have been devised to describe the different levels of organization of

networks and to measure their degree of order. Among the different measures used with this aim, *entropy* (as a measure of disorder) holds a prominent place. In a recent paper, Bianconi (2009) defined and characterized the notion of *structural entropy*, i.e., the entropy of ensembles of undirected, uncorrelated, simple networks with given degree sequence. She pointed out the apparent paradox that scale-free degree distributions are characterized by having a small structural entropy even though they are frequently encountered in the real world. She explains this finding by noticing that scale-free networks correspond to the most likely degree distribution given the corresponding value of the structural entropy.

Random Boolean networks

Random Boolean networks have been recently investigated by Andrecut and Kauffman (2010). These networks are non-linear and show a well-understood transition between ordered and disordered phases. The authors studied their complex dynamics as a function of the connectivity k between the elements of the network. They have uncovered an order-chaos phase transition for a critical connectivity $k_{cr} = 2$; moreover, pairwise correlation and complexity measures are maximized in dynamically critical networks. Their results are in good agreement with the previously reported studies on random Boolean networks and random threshold networks.

Percolation

A phase transition corresponding to explosive percolation has been found by Friedman and Nishimura (2010) in several models of random networks, widely applicable to physical, biological, and social networks. In such networks giant clusters appear (essentially) without warning, phenomenon that may have important implications in several fields.

Finally, Komin and Toral (2010) have used ideas from statistical physics to reduce large systems of coupled differential equations with diverse parameters to three equations, one for the global mean field variable and two describing the fluctuations around this mean value. With this tool they analyzed phase transitions induced by microscopic disorder in three prototypical models of phase transitions, which have been studied previously in the presence of thermal noise. Macroscopic order is induced or destroyed by time-independent local disorder. A finite-size analysis of the numerical results allows the calculation of the corresponding critical exponents.

Small-world networks

Transitions in networks from a small-world structure to a fractal structure have been investigated by Rozenfeld *et al.* (2010). By using the renormalization group they showed that network topologies can be classified into universality classes in the configuration space. Moreover, they found a trivial stable fixed point of a complete graph, a non-trivial point of a pure fractal topology (stable or unstable according to the amount of long-range links in the network), and another stable point of a fractal with shortcuts that exist exactly at the small-world to fractal transition. In addition, they were able to explain the coexistence

of fractal and small-world phases and to extract information on the distribution of shortcuts in real-world networks.

12.2 Statistical physics and the social sciences

Complex networks have found an increasing number of applications in the social sciences, where both classical and statistical mechanical methods have been applied to investigate their properties. A comprehensive overview of statistical mechanics methods used to model and study social systems was provided by Castellano *et al.* (2009). These authors worked on opinion formation, cultural dynamics, collective behaviors, and the coevolution of topology. Social networks, notwithstanding the limited number of interactions that people may have, show remarkable ensemble behaviors, such as transitions from disorder to order. An example of this is the spontaneous formation of a common language/culture. There are also cases of scaling and universality.

The idea of modeling social phenomena using physics tools is not new, as it was already present in the thought of philosophers such as Laplace and Comte. However, only in the past few years has the idea moved from a philosophical possibility to concrete research involving many physicists. This was motivated, on the one hand, by the availability of large networks for study and, on the other, by new social phenomena (mostly related to the Internet) observable on a large scale.

Social dynamics

As Castellano *et al.* (2009) noticed, a conceptual difficulty immediately arises when one is approaching social dynamics from the point of view of statistical physics. In fact, in physics, elementary constituents of complex systems are simple objects whose behavior is well known. Thus, the observed macroscopic phenomena derive substantially from the complexity of the interaction and not from the complexity of the constituents. With humans, modeling involves strong simplifications that are not always well grounded; any investigation of models of social dynamics involves an additional difficulty, namely the very definition of realistic microscopic models of the entities involved (humans, in particular). However, many macroscopic properties do not depend on the details of the component processes, and qualitative results at least can be obtained that match the observed reality. Hence, in many cases it is sufficient to include in the model only the basic properties necessary to describe an individual behavior.

As a general finding, in social models the drive toward order is provided by the tendency of the agents to become more alike. This effect is termed *social influence* and can be seen as an analogue of the ferromagnetic interaction in magnets. Even though analogies can be drawn between social systems and statistical physics systems, there are concepts that present more difficulty in physical

interpretation. For instance, in many social dynamic models there is the idea that only quite similar agents are willing to interact. This concept is not immediately translatable into statistical physics but can be loosely associated with the concept of the distance between particles. A fundamental aspect of social modeling is diversification in the topology of interactions whereas, in statistical physics, patterns of interaction are often regular lattices. On the contrary, more plausible interaction patterns are those determined by more complex topologies, such as small-world or scale-free ones.

12.2.1 Opinion dynamics

In *opinion dynamics* the dynamics of agreement or disagreement between individuals is complex. In statistical physics approaches, the opinion states of a population are defined as well as the elementary processes that determine transitions between such states. One of the first pieces of work using statistical physics methods in opinion dynamics was by Galam *et al.* (1982), who applied the Ising model with spin-spin coupling representing the pairwise interaction between agents and the magnetic field representing the cultural majority. Personal preferences toward either orientation were modeled as individual fields. Depending on the strength of the individual fields, the system may reach either total consensus toward one of two possible opinions or a state where both opinions coexist.

Voter model

One of the main approaches to opinion formation is the *voter model*. Voter dynamics was first discussed by Clifford and Sudbury (1972) as a model for the competition of species. It reached popularity because it is one of the very few non-equilibrium stochastic processes that can be solved exactly in any dimension. In the model, each agent is modeled by a binary variable $s = \pm 1$. At each time step an agent i is selected along with one of its neighbors j and s_i is set equal to s_j, i.e., the agent takes on the opinion of the neighbor. This updating rule implies that agents tend to imitate their neighbors. Starting from a disordered initial condition, voter dynamics tends to increase the order of the system.

Recently, more complex models have been proposed. For instance, Castelló *et al.* (2006) introduced the *AB* model with three states. At each time step an agent i is randomly chosen and its state is updated according to the following transition probabilities:

$$\mathtt{P}_{A\to AB} = \tfrac{1}{2}\sigma_B, \qquad \mathtt{P}_{B\to AB} = \tfrac{1}{2}\sigma_A,$$
$$\mathtt{P}_{AB\to B} = \tfrac{1}{2}(1-\sigma_A), \qquad \mathtt{P}_{AB\to A} = \tfrac{1}{2}(1-\sigma_B).$$

where σ_k ($k = A, B$) is the local density of each state in the neighborhood of i. The idea here is that in order to go from A to B one has to pass through the

intermediate state AB. The rate for going from state A to AB is proportional to the density of neighbors in state B. This implies that consensus on the AB state, or a frozen mixture of A and B, is not possible, the only two possible absorbing states being those of consensus of either the A or the B type.

Non-regular topologies have important effects on the dynamics of the voter model. On a complete graph the one-dimensional diffusion equation, with a position-dependent diffusion constant, can be solved analytically.

The average time needed to reach consensus in a system of finite size can be computed exactly for any value of the initial "magnetization" and increases with the size of the system. When considering disordered topologies different ways of defining the voter dynamics, which are equivalent on regular lattices, are nonequivalent generalizations of the voter model. When the degree distribution is heterogeneous, the order in which a site and the neighbor to be copied are selected is relevant because high-degree nodes are more easily chosen as neighbors than low-degree ones.

An interesting effect of the topology occurs on small-world networks. After an initial regime the system remains trapped in a metastable state with coexisting domains of opposite opinions (Vilone and Castellano, 2004). The lifetime of the metastable state is linear with the system size, so that for finite systems consensus is eventually reached on a temporal scale shorter than that for a regular one-dimensional lattice. For infinite systems the state with coexisting opinions is actually stable, leading to the conclusion that long-range connections prevent a complete ordering of the voter model.

Majority rule model

An alternative approach to opinion dynamics is the *majority rule* (MR) model. A population includes N agents, such that a fraction p^+ of agents has opinion $+1$, while a fraction $p^- = 1 - p^+$ has opinion -1. For simplicity, it can be assumed that the interconnection network is a complete graph. At each iteration a group of r agents is selected at random (a discussion group): they interact and finally all agents assume the majority opinion inside the group (Galam, 2002). The group size r is not fixed but is selected at each step from a given distribution. If r is odd, there is always a majority in favor of either opinion. If r is even and it happens that either opinion is supported by exactly $r/2$ agents, bias is introduced in favor of one of the opinions, say $+1$, and then that opinion prevails in the group.

Social impact theory

Lewenstein *et al.* (1992) proposed a model of social impact that can be solved in the mean-field approximation. A population contains N individuals i, each characterized by an opinion $\sigma_i = \pm 1$ and by two random parameters, the *persuasiveness* p_i and the *supportiveness* s_i, which describe the capability to convince an

individual to change or to keep its opinion, respectively. These two parameters represent the quenched disorder of the system. Let I_i be the total impact that an individual i experiences from its social environment; then the opinion dynamics is expressed by the rule:

$$\sigma_i(t+1) = -\mathrm{sgn}[\sigma_i(t) I_i(t) + h_i],$$

where h_i is a random field representing all sources other than social impact that may affect the opinion. According to the above rule, an opinion flips if the pressure in favor of the opinion change overcomes the pressure to keep the current opinion. In general, in the absence of individual fields, the dynamics leads to the dominance of one opinion on the other but not to complete consensus. If the initial "magnetization" is about zero, the system converges to configurations characterized by a large majority of "spins" in the same opinion state, and by stable domains of "spins" in the minority opinion state. In the presence of individual spin fields these minority domains become metastable: they remain stationary for a very long time and then suddenly shrink to smaller clusters, which again persist for a very long time before shrinking again, and so on ("staircase dynamics").

12.2.2 Social and cultural dynamics

Cultural dynamics is the multivaried version of opinion dynamics. The basic question is related to the uncovering of the microscopic mechanisms that drive the formation of a cultural domain and of its eventual persistence. The work of Axelrod (1997) has been very influential; he included in the model two mechanisms that are believed to be fundamental in the understanding of the dynamics of cultural assimilation (and diversity): *social influence* and *homophily*. The first is the tendency of individuals to become more similar when they interact. The second is the tendency of likes to attract each other, so that they interact more frequently. These two ingredients were generally expected by social scientists to generate a self-reinforcing dynamics leading to a global convergence to a single culture. It turns out instead that the model predicts in some cases the persistence of diversity.

From the point of view of statistical physics, the Axelrod model is a simple vectorial generalization of models of opinion dynamics that generates some truly novel behavior. In the model, individuals are located on the nodes of a network and are associated with F integer variables $(\sigma_1, \ldots, \sigma_F)$ that can assume q values $\sigma_f = 0, 1, \ldots, q-1$. The variables are called *cultural features* and q is the number of the possible traits allowed per feature. They are supposed to model the different "beliefs, attitudes and behavior" of individuals. In an elementary dynamic step an individual i and one of its neighbors j are selected, and the

overlap between them, namely

$$\omega_{i,j} = \frac{1}{F}\sum_{f=1}^{F}\delta_{\sigma_f(i),\sigma_f(j)},$$

is computed. With probability $\omega_{i,j}$ a feature for which traits are different ($\sigma_f(i) \neq \sigma_f(j)$) is set equal to $\sigma_f(i)$. Otherwise there is no change. Clearly the dynamics tends to make interacting individuals more similar, but interaction is more likely for neighbors already sharing many traits (homophily) and becomes impossible when no trait is the same. There are two stable configurations for a pair of neighbors: when they are exactly equal, so that they belong to the same cultural region, or when they are completely different, i.e., they are on either side of the border between cultural regions. Starting from a disordered initial condition, the evolution of any finite system leads unavoidably to one of the many absorbing states, either a state in which all individuals have the same set of variables or a frozen state with the coexistence of different cultural regions. The state reached depends on the number of possible traits q in the initial condition. For small q, individuals share many traits with their neighbors and full consensus is quickly achieved. For large q, very few individuals share traits and there is the emergence of small cultural domains that are not able to grow, a disordered frozen state. On regular lattices the two regimes are separated by a phase transition at a critical value q_{cr}, which depends on F.

Recently, Contucci *et al.* (2008) applied statistical mechanics models to describe the cultural interchange between two homogeneous groups of individuals. The interaction inside a group is *imitative* whereas across groups it may be either *imitative* or *counter-imitative*. When two populations come into contact, as in the case of immigration but also in a more general context through the media, sometimes cultural traits are evenly mixed and sometimes one population's traits dominate. An interesting finding is that, in some cases, the changes vary smoothly with the relative proportions of the two groups whereas in other cases the crossing of a critical value triggers a jump in the observed quantity (Michard and Bouchaud, 2005).

Contucci *et al.* (2008) built a mean field theory of the two-population problem, i.e., they assumed that every individual interacted with every other with the same strength. To build their model, the authors mapped the resident–immigrant cultural interaction problem onto that of two interacting groups of spins. A binary spin $\sigma_i \in \{+1, -1\}$ is associated with each cultural trait. The interaction between two individuals i and j is modeled by a *potential*, which reflects the will to agree or disagree between the two. The system has two control parameters: α, the ratio of the number N_1 of immigrants and the total cardinality N of the population, and J_{12}, which represents the strength of imitation or counter-imitation across the two groups. The output is the average *magnetization* $\langle m \rangle$, which, in

this context, represents the average opinion of the interacting system. The obtained results show that $\langle m \rangle$ varies smoothly when the interchange coefficient is small but abruptly when the coefficient is large. More intriguing is the observed dependence of the critical percentage on the internal cohesion of each group. Owing to a fine balance between internal energy and entropy they found that a strong cohesion penalizes the cohesion of group.

Another study that revealed the presence of a phase transition in a social system was performed by Woloszyn *et al.* (2007). Their goal was to investigate the social ability to become organized, as a function of the topology of a socialties network organized into *cliques*. Again, a spin σ_i is assigned to each node and an interaction energy J to each link. The authors' aim was to discover the presence of a phase transition; if it exists, it is indeed a sign of the ability of the network to become organized. The results indicate that if the connections between the small groups are too sparse then the system as a whole does not show any collective behavior.

12.3 Communication and computation networks

Another type of complex system for which the methods of statistical physics are particularly well suited are large networks of communicating devices, such as telephone networks, computational grids, the Internet, and the World Wide Web. In recent years, in fact, these networks have reached sizes that justify ensemble approaches to their analysis. It is to be expected, then, that emerging phenomena such as phase transitions are found in their behavior.

Ohira and Sawatary (1998) created a simulation model of traffic in a computer network and found the presence of a phase transition. They were particularly interested in the shifting of the phase transition points from a state of low congestion to one of high congestion when the strategy for selecting paths for the packets is changed. The order parameter of the transition was the average travel time of packets and the control parameter was the generation rate of packets in the network. The interest of the work is that the authors proposed a stochastic routing strategy that shifts the location of the phase transition. If a parameter, used to control randomness in the path selection, assumes an appropriate value, then the onset of the congestion phase (at the phase transition) is delayed, i.e., the network can accomodate a higher rate of packet generation before becoming congested. The authors also changed the number of routers that apply this strategy and found that the model shows a non-linear response as a function of the proportion of these routers. This observation suggests that the phase transition is mostly due to their interaction.

Lawniczak and Tang (2005) investigated a similar problem, namely the packet traffic dynamics in a data network model near the phase transition point

from free flow to congestion. They built a model to investigate the spatiotemporal dynamics of packets traffic near the phase transition point for different network connection topologies and for static or adaptive routing algorithms. The results showed, for adaptive routings and periodic square-lattice connection topologies, the emergence of synchronization, i.e., of a structure with peaks and valleys in the distributions of outgoing queue sizes when the network models are in their congested states. The emergence of this type of synchronization is accelerated by adding an extra link but destroyed by adding many more links. With adaptive routings and periodic triangular-lattice connection topologies, the packet traffic was much more evenly distributed.

In a more recent paper Marbukh (2007) discussed the possible presence of a phase transition in various types of complex communication networks, as well as the consequences of these phenomena for network performance evaluation and control. The microscopic description of the network was given by a Markov process with a large number of locally interacting components. The relation between microscopic and macroscopic descriptions was studied using statistical physics tools.

A more comprehensive approach to the analysis of synchronization in complex networks, including both numerical and analytical approaches, was provided by Arenas *et al.* (2008). The authors offered explanations of the synchronization occurring in a network with complex topology when oscillating elements interact. Moreover, they highlighted the new features emerging from the interplay between the structure and the underlying pattern of connections. The investigation was extended to the analysis of opinion formation, where numerical simulations show that there is a phase transition from incoherence to synchrony at a well-defined critical coupling.

A directed small-world network consisting of attractively coupled identical phase oscillators has been analyzed recently (Tönjes *et al.*, 2010). The authors found that complete synchronization is always stable but is not always reachable from random initial conditions. Depending on the shortcut density and on the asymmetry of the phase coupling function, there exists a regime of persistent chaotic dynamics. On increasing the density of shortcuts or decreasing the asymmetry of the phase coupling function, they observed a discontinuous transition in the ability of the system to synchronize.

12.4 Biological networks

Large biological networks, such as, for instance, gene regulatory networks, are typical of the complex systems that can be found in many branches of biology and physiology. Problems in these areas have been often approached via

machine learning techniques, both supervised and unsupervised; cluster analysis, in particular, has been widely applied, for example to find groups of related genes (Naczar *et al.*, 2007). Clearly, such large networks have also been investigated with complex systems tools with aim of linking biological properties to network measures, such as *betweenness*, *centrality*, *degree distribution*, and so on.

Very recently, methods from statistical mechanics have been used also with very promising results. One approach was proposed by Braunstein *et al.* (2008), who analyzed and compared two different algorithmic approaches to identify gene-regulatory interactions from high-throughput gene expression data. The first approach uses pairwise correlations between regulated and regulating genes and the second uses message-passing techniques for inferring activating and inhibiting regulatory interactions. The message-passing (belief propagation) techniques can be understood as an algorithmic reinterpretation of the cavity method in spin glass physics. It is the same idea as that underlying the survey propagation algorithm described in Section 3.5.4. The performances of the two algorithms have been analyzed theoretically on well-defined test sets, using tools from the statistical physics of disordered systems such as the replica method. The message-passing algorithm was found to outperform the classical algorithms, since it takes into account the collective effects of multiple regulators.

An interesting work linking evolutionary learning to gene network generation appeared recently (Nicolau and Schoenauer, 2009). The authors proposed a novel approach to generating scale-free network topologies that is based on an existing artificial gene-regulatory-network model. From this model different interaction networks can be extracted, on the basis of the value of an activation threshold. Using an evolutionary computation approach, the model is allowed to evolve in order to reach network-specific statistical measures. The results obtained show that, when the model uses a duplication and divergence initialization, such as that seen in nature, the resulting regulation networks are closer in topology to scale-free networks. Indeed, these initialized genomes are far better suited for evolution than are purely random networks, owing to the larger range of degrees in the networks they encode as well as to the wider choice of resulting networks obtained by varying the threshold parameter that decides the existence of an edge between nodes.

12.5 Comments

The networks mentioned in this chapter have different natures regarding their composition at the microscopic level. In fact, the component elements differ greatly in their complexity, autonomy, and richness of interactions, ranging from

proteins and computers to human beings. It is then quite striking that analogous patterns of behaviors appear.

As a matter of fact, once the components have been brought to a suitable level of abstraction (keeping only the essentials of their individual behavior and discarding irrelevant details) only the pattern and strength of their interactions matters. It is thus possible to find commonalities in systems that differ largely in their composition but not in their behavior, allowing the transfer of results from one domain into another.

For this reason, this chapter does not aim at providing full details of the approaches and results but suggests where one should start to look for similarities in other domains.

13

Phase transitions in natural systems

Contents

Ensemble phenomena such as the emergence of a phase transition are by no means limited to artificial and inanimate entities. On the contrary, in living beings one may not infrequently observe similar phenomena, whose origin can be tracked down to the effect of the interaction among many components. Indeed, in living organisms the number of cells cooperating in biological and physiological processes is so large that ensemble behaviors are only to be expected. The rapid transition between two neural states was observed as early as the late 1980s (Freeman, 1988). The experimental setting used by Freeman and co-workers included animal and human subjects, engaged in goal-directed behavior, on which high-density electrode arrays were fixed. Action potentials and brain waves (electroencephalograms (EEG) and local field potentials) were then recorded and used to develop a data-driven brain theory.

Freeman and co-workers designed data processing algorithms that enhance the spatial and temporal resolution of the textures of brain activity patterns found in three-layer paleocortex and six-layer neocortex. A major discovery of their work was evidence that cortical self-organized criticality creates a pseudo-equilibrium in brain dynamics that allows cortical mesoscopic state transitions to be modeled analogously to phase transitions in near-equilibrium physical systems. In more detail, the uncovered transition has four stages: a *first-order phase transition*, which resets the phase of beta–gamma EEG oscillations in a

discontinuity of the cortical dynamics; a *pattern selection* in the attractor landscape by phase resynchronization; a *second-order phase transition*, which leads to pattern stabilization by a dramatic decrease in the rate of change of the order parameter; and then *high-energy pattern broadcast* over divergent–convergent axonal cortical output pathways, during which the rate of free energy dissipation is maximized. The ratio of the rate of free energy dissipation and the rate of change in the order parameter defines the pragmatic information, which is maximized during cortical transmission. On the basis of these findings the authors noticed that the power law and fractal distributions of EEG parameters enabled the scale-free dynamics of the cortex to be displayed as a macroscopic cortical state transition that sometimes covers an entire cerebral hemisphere almost instantaneously, even in humans; thus, they hypothesize that this transition is the neural mechanism that forms *Gestalts* (unified multisensory percepts).

Freeman also investigated the emergence of a phase transition in the animal visual system (Freeman, 1990). By analogy with the olfactory system, he suggested that in the visual cortex also a state transition causes a jump in the cortical dynamic state, constituting a type of bifurcation. The model proposed by Freeman requires that the cortex be intrinsically unstable and liable to sudden transitions under the appropriate stimuli. The conditions that facilitate controlled instability include a high level of cortical activity and of excitability, which is achieved under the neurochemical states of behavioral arousal and motivation. In a suitably aroused animal that expects a certain stimulus, the arrival of the sought stimulus can induce neural activity that serves as a bifurcation parameter.

After two decades of work on the subject, the presence of phase transitions in perception is still at the core of Freeman's interest. In a recent paper (Freeman, 2009) he investigated the process of storing information, extracted from microscopic sensory inputs, in the mesoscopic memory for retrieval in recognition. The process requires the creation of spatio-temporal patterns of neural activity. Such a construction occurs through phase transitions in cortical populations that condense the background activity through spontaneous symmetry breaking. Large-scale interactions create fields of synaptically driven activity, which is observed by measuring brain waves (with an electrocorticogram) and evaluated by constructing a mesoscopic vectorial order parameter.

Recently, Cowan also investigated in depth the question of phase transitions in the brain (Koppes, 2008). According to his view the same rules that are valid in the transition from vapor to liquid or from liquid to solid can be applied to the activation schemes followed by the neurons in the human brain. In particular, he observed the emergence of phase transitions as a consequence of neural interactions: he showed that, using the mathematical tools provided by statistical physics, it is possible to explain how the rhythms observed with an electroencephalogram, including δ-waves (occurring during sleep), α-waves (linked to visual processing), and γ-waves (linked to information processing), are

generated. Cowan was a proposer of the *Wilson–Cowan equations*, which are aimed at describing the dynamic activity of (biological) neural nets. Even though these equations are too simple to be realistic, they are nevertheless a first step towards relating neural phase transitions and neurological conditions or cognitive states.

Neurological functioning in humans is not the only such area where phase transitions have been found. For instance, Szabo *et al.* (2006) recorded the swarming-like collective migration of a large number of keratocytes (tissue cells obtained from the scales of goldfish) using long-term videomicroscopy. The authors showed that on increasing the density of the migrating cells, a kinetic phase transition from a disordered to an ordered state occurs. Near the critical density, interacting clusters of cells, moving in groups, appear. On the basis of these experimental results they proposed a flocking model that exhibits a continuous transition to the ordered phase, assuming only short-range interactions and no explicit information about the directions of motion of neighbors. Placing cells in microfabricated arenas, they found a spectacular whirling behavior, which they could also reproduce in simulations. The formation of groups of moving cells near the phase transition is surprisingly similar to the formation of clusters of solutions in the analogous SAT problem, as described in Section 3.3.2.

Another piece of work related to the emergence of a phase transition was reported by Kshivets (2008). He found that in lung cancer the diameter of the tumor cell shows a critical value (2 cm) that separates a state in which the five-year survival chance after surgery is 100% from a state where the survival chance falls sharply. It was earlier proved that this happens because there is a phase transition in so-called "early-invasive lung cancer" at a critical level of the lung-cancer cell population.

Carmesin and Arndt (1995) described a neural network, constituted by sensors (the input layer) and inner neurons (the hidden layer), that models multistable visual perception (Kruse *et al.*, 1996). The authors proposed a dynamic model that includes a stochastic neuronal dynamics, a formal Hebb-type coupling dynamics, and a resource mechanism that corresponds to saturation effects in perception. The model comprises a set of differential equations. Single stimuli are bound to exactly one percept, even in ambiguous situations where multistability occurs. The network exhibits both discontinuous and continuous phase transitions and can model various empirical findings, including the percepts of succession, alternative motion, and simultaneity; the percept of oscillation is explained by the oscillation of percepts at a continuous phase transition. In particular, Carmesin and Arndt studied the phenomenon of stroboscopic alternative motion. Increasing the frequency of presentation, there are five different percepts: (a) succession, (b) fluttering motion, (c) reversible clockwise and anticlockwise turning motion, (d) oppositional motion, and (e) simultaneity. The frequency of presentation serves as the control parameter, whereas the percepts

are the order parameters, of the phase transition. Both theoretical prediction and experiments agree on the existence of the three phases (a), (c), and (e), with the two continuous phase transitions (b) and (d) in between.

Work in a related field was done by Kelso *et al.* (1990) and Fuchs *et al.* (2000). These authors studied the transition in coordination behavior from *syncopation* to *synchronization* in human subjects. The subjects' task was to perform a flexion movement of the preferred index finger in between two consecutive tones of an auditory metronome, i.e., to syncopate with the stimulus. It is well known that, by increasing the presentation rate of the stimuli as a control parameter, a point is reached where subjects can no longer perform a stable syncopated coordination pattern and, under the instruction to keep pace with the metronome, switch spontaneously to a movement that is instead synchronized with the stimulus. Three major changes in brain activity take place when the switch in the movement behavior occurs.

- The topography of the dominant spatial pattern of brain activity changes.
- The frequency of the time-dependent amplitude of neuromagnetic activity corresponding to the pattern described above switches from the coordination frequency (prior to the transition) to twice the coordination frequency (after the transition).
- In certain sensors the time series undergoes a phase shift of π at the same time as the transition in the coordination behavior.

Subsequent theoretical work established the nature of the phase transition at both brain and behavioral levels through phenomenological modeling. More recently, a theory connecting the brain and behavioral levels of description has been developed, based on the known cellular, and neural, ensemble properties of the cerebral cortex.

Another phenomenon in which a phase transition has been uncovered is in the trace conditioning paradigm, specifically in the air-puff eye-blink paradigm (Howe and Levy, 2007). The paradigm consists of presenting to rabbits two temporally separated non-overlapping stimuli (air puffs) with a specified amount of stimulus-free time in between. A rabbit should learn to anticipate the second air puff, the *unconditioned stimulus* (US), by a timely blinking just prior to its onset. The US follows at a specified time after the offset of a *conditioned stimulus* (CS). The stimulus-free time between the CS offset and the US onset is called the *trace interval*. As the paradigm requires the rabbit to predict the US onset based on the CS, the paradigm belongs to the class of problems handled by the authors' computational theory of the hippocampus as a multisensory sequence encoding and predicting system. The ability to predict the US onset shows an

abrupt change. No US neurons predict the US for 95 trials. Then, within five additional trials, more than 30% of US neurons produce a timely prediction. After a training period, characterized by a failure in prediction, rabbits suddenly and accurately begin to predict the US onset. Moreover, a prior investigation of CS and US longevity noted a phase-transition-like behavior in the predictive mode that was dependent on the trace-interval length (Wu and Levy, 2005).

An overview of phase-transition-like phenomena in brain functioning was reported by Werner (2007), who considered both experimental observations and theoretical models. The brain clearly appears to be a non-linear system operating at the edge of criticality, which is achieved and maintained by self-organization. The concepts of scaling and universality, derived from statistical physics, prove to be useful notions for explaining the nature of the underlying neural processes occurring in neural circuits of cerebral cortex and subcortical structures. A similar view was proposed by Haken (2002), who considered the brain as a pattern-forming system that operates close to instability in order to achieve flexible and rapid switching between coherent states. Basar contributed to this view of the brain as a dynamic system as early as the 1980s (Basar, 1983). Since 1975, Freeman has produced a steady flow of studies of the dynamic principles of wave patterns in brains, which have yielded numerous relevant findings including characterizations of attractors, bifurcations, and critical phase transitions (Freeman, 1975; Freeman and Vitiello, 2006).

Recently, Cocco *et al.* (2009) have applied methods from *inverse* statistical physics to infer coupling between retinal ganglion cells in salamanders. As the complexity of neural systems often makes it impractical to measure the interactions between neural cells, the authors propose to use inverse statistical physics approaches to infer effective couplings between neurons from their spiking activity. In particular, they described two computationally efficient inverse algorithms, based on the Ising and "leaky integrate-and-fire" models, and applied them to re-analyze multielectrode recordings in the salamander retina in darkness and under random visual stimuli. The authors found strong positive couplings between nearby ganglion cells common to both types of stimulus, whereas long-range couplings appear under random stimulus only. They claimed that the proposed methods would also allow the real-time evaluation of couplings for large assemblies of neurons.

13.1 Comments

Given the internal complexity of biological entities, it should be expected that ensemble phenomena will occur. In fact, most activities of life consist of the interaction of large numbers of small components, be they proteins in regulation

networks, cells in organs, or neurons in the brain. However, only recently have these patterns of interactions become the object of investigation with methods derived from complex systems theory and statistical physics.

Even though *natura non facit saltus*, according to Carl von Linnée[1] (and Leibniz and Darwin), we do experience discontinuities, especially in our cognitive functioning. For instance, this happens with *ambiguous illusions*, which are pictures or objects that may have two valid interpretations that are not both visible at the same time: the viewer can only switch from one to another, as if there is a barrier to cross. In reasoning also we may suddenly reach a long-sought goal, such as grasping or learning a concept, after a critical amount of information has been collected. In the future this situation might be related to what happens in artificial neural networks, reported in Section 7.1. In consequence the investigation of cognitive processes with methods from statistical physics may prove to be both viable and successful.

[1] *Philosophia Botanica*, Stockholm, 1751.

14

Discussion and open issues

Contents

In the long journey undertaken in this book, we have visited statistical mechanics, constraint satisfaction problems and satisfiability, complex networks and natural systems, and, in particular, many facets of machine learning ranging from propositional to relational learning, grammatical inference, and neural networks. The thread that connects all these fields is the emergence of phenomena exhibiting sharp discontinuities. These phenomena are reminiscent of the phase transitions found in physics and, indeed, the methods of statistical physics have been employed with success to analyze them. In this chapter we try to summarize what we have learned from these connections and in particular from the role played by machine learning. Our aim is to point out gaps in the understanding of basic phenomena and to identify open questions that may suggest future research directions.

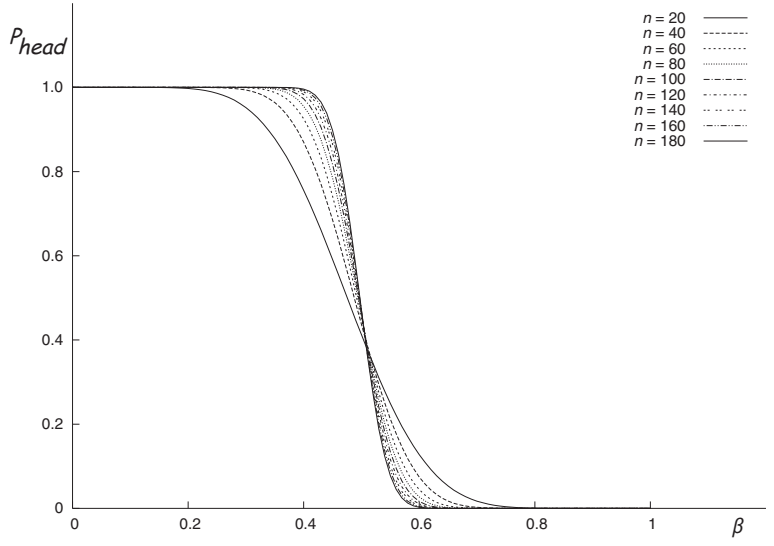


Figure 14.1 Probability P_{head} vs. β for various values of the number n of trials in a single session.

14.1 Phase transitions or threshold phenomena?

In a recent and very interesting paper, which recalls similar arguments put forwards in Percus *et al.* (2006), Zweig *et al.* (2010) have challenged the current view of phase transitions in computational problems, wondering whether the abrupt change observed in the probability of solution (the *order* parameter) in SAT problems is in fact nothing other than a "self-fulfilling" discontinuity, i.e., an *existential* discontinuity generated by the very definitions of the problem and of the order parameter.

The first argument in support of their claim is that it is easy to produce rather simple models that exhibit phase transition phenomena while, as most of us would agree, the essential ingredients that underly a "true" phase transition are lacking. Indeed, Zweig *et al*. managed to produce a model that tightly fits the measures available in the 3-SAT problem.

Coin tossing

Regarding the models exhibited by these authors, the first consists in *tossing a coin* n times with a probability $1 - \beta$ of outputting *head*. The probability β is the control parameter. The order parameter is the probability P_{head} that among the n tosses there is a majority of *heads*. Clearly, when $n \to \infty$, $P_{head} \to 1$ below $\beta = 0.5$ and $P_{head} \to 0$ above. The system also shows a finite size-scaling effect, with exponent 0.5. In Figure 14.1 P_{head} is shown as a function of

β for various values of the number of trials n. The increasing sharpness with n of the curves in Figure 14.1 seems to be ascribable to the law of large numbers rather than a transition between two "phases".

Coupon collector problem

The other problem is the *coupon collector* problem: there are n distinguishable items (the "coupons"), named "1", ..., "n", and multiple copies of each are contained in a large multiset S. A collector extracts an item from S one at a time, uniformly at random, and puts it in his or her collection C, if it is not yet present there. Let C_k be the collection when it contains k different items. The collection is complete when $k = n$. As S is very large, extraction with and extraction without replacement are almost equivalent and the probability of extracting a new item, not already belonging to C_k, can be written as $(n - k)/n$. The expected number of ways y_k of obtaining C_{k+1} from C_k is given by the equation

$$y_k \, \frac{n-k}{n} = 1,$$

i.e., $y_k = n/(n-k)$. The total number of ways y of obtaining C_n is then

$$y = \sum_{k=0}^{n-1} \frac{n}{n-k} = nH_n,$$

where H_n is the nth harmonic number. Let us now generalize the problem from just one to x collections, and let $P_{full}(t)$ be the proportion of full collections after t draws. By taking into account the size n of the problem, and by introducing the Euler–Mascheroni constant $\gamma = 0.577$, a variable τ is defined:

$$\tau = \frac{t}{nH_n} \simeq \frac{t}{n \ln n + \gamma n + 0.5}.$$

Plotting $P_{full}(\tau)$ versus τ gives a graph similar to that in Figure 14.1, with an apparent phase transition at the value $\tau_{cr} = 1$. Also, the system shows a non-trivial finite-size scaling effect with exponent 0.17.

True phase transition?

On the basis of the two simple examples reported above, Zweig *et al.* (2010) observe that, notwithstanding their mathematical behavior, our intuition hardly accepts that these systems undergo a "true" phase transition. And this observation leads them to raise the issue of a precise definition of what a phase transition really is. Informally, in their opinion, a "true" phase transition should correspond to some structural and deep change in the system under study that is independent of the definition of the order parameter. In other words, the "phases" must be defined first and then the nature of the transition between them has to be investigated as opposed to defining the phases from the behavior of the order parameter itself (i.e., the phases are defined as regions of the control parameter space where the order parameter assumes different values). This is what happens in physical

systems when, for instance, water freezes or boils. Even though contributing to this definitional discussion is beyond the scope of the book, we will make some comments on the implications that such discussion may have on our problem of interest, namely relational machine learning.

Before proceeding any further it is important to realize that whatever the true nature of the phase-transition-like phenomena at play in machine learning, their impact on the practicality of machine learning is undiminished. However, we agree with intuition of Zweig *et al*. that in a phase transition phenomenon something more fundamental should be happening than the typical mathematical behavior of the order parameter. In particular, we believe that two essential aspects should be present:

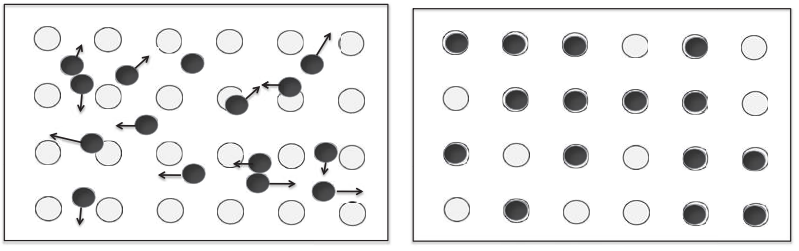
1. the existence of a *micro* level where a large number of entities are present;
2. some interaction of these entities that produces ensemble phenomena (such as a phase transition) at the *macro* level.

The above is also the context in which statistical physics methods are applicable. Whereas in neural networks, in complex social and information networks, and in the brain the micro and macro levels are quite easy to identify (as we saw in Chapters 7, 12, and 13, respectively), in computational problems such as SAT and symbolic machine learning it may be more difficult to do this.

Considering the coin-tossing problem, clearly neither condition 1 nor condition 2 above is verified; in fact, there is no underlying structure, and each toss is independent of the others, so that no interaction exists. Extrapolating from this example and from the coupon collector example, one may wonder whether a similar threshold phenomenon on a stochastic variable might actually be the mechanism underlying several other apparent phase transitions. However, as suggested above, the answer to this question may actually be irrelevant as the important thing is the effect that this type of behavior has in practice. To be more precise, the findings that emerged in Chapter 10 set limits on the feasibility of relational machine learning, independently of the actual nature of the transition investigated in Chapter 9. Likewise, for k-SAT and other $\mathbf{NP}$-hard problems it is relevant in practice to know where the most difficult instances are located in the space of the control parameters. Nevertheless, to distinguish between a true phase transition and an apparent one has a significant impact on a deep understanding of the problem under study. Therefore, the need for a precise definition of a phase transition depends on the perspective one takes.

Feedback

Before moving to machine learning we would like to illustrate a further point with an intuitive example, namely, the idea of *feedback* in complex sets of interacting entities. Let us consider the toy system in Figure 14.2, which shows a box whose bottom is covered by a regular grid of cylindric holes. Suppose that a some pellets randomly slide over the bottom of the box, colliding with each other and



(a) (b)

Figure 14.2 Toy model for a complex system exhibiting a phase transition. (a) When the pellet speed is above a given threshold the pellets fly over the holes. (b) When the average speed decays below a given threshold, the pellets are quickly entrapped in the holes. The arrows in (a) indicate the pellets' vector velocities.

with the box walls, according to a pseudo-Brownian motion. More specifically, let us assume that the velocity v_i of pellet π_i follows a normal distribution with average value v_{avg} and standard deviation σ (the motion is isotropic). Finally, around the rim of each hole is a small incline, which imparts a vertical velocity to the pellets.[1] As long as the pellets have a speed whose modulus is above a given threshold v_{cr}, they fly over the holes and the motion continues.[2] Only those pellets whose velocity lies below v_{cr} fall into the holes. Their number depends on σ. When the average speed decays below the threshold, more and more pellets fall into the holes. If we define as an order parameter the proportion of pellets that have fallen into holes and as a control parameter the average speed v_{avg}, the order parameter decreases with increasing v_{avg}. When $v_{avg} = v_{cr}$ the proportion of fallen pellets is 0.5. The form of the function is typical of a phase transition and its steepness increases with decreasing σ. In fact, the function becomes a step function when $\sigma = 0$.

This case is analogous to the coin tossing problem, as each pellet is effectively independent of the others as long as their number allows a macroscopic description of their motion. On the contrary, the number n of pellets, if sufficiently large, does not influence the results unless we consider the normal distribution as a description of the fluctuations around the mean velocity. In this case, the intensity of the fluctuations decreases as $1/\sqrt{n}$, approaching 0 for increasing system size. Even though the ensemble of pellets may constitute a microscopic level for the system, the elementary constituents do not interact so that it is still

[1] This is necessary otherwise the pellets would always fall into the holes whatever their horizontal speed.

[2] The threshold v_{cr} is a constant depending on the system geometry (the hole size), the vertical velocity, and the elasticity, texture, etc. of the material used to construct the box and balls.

difficult to consider the introduced order parameter as revealing a phase transition. In fact, as in the case of the coin tossing, the transition seems dictated by the law of large numbers.

Now, let us complicate the system a little more, by assuming that a pellet that has fallen into a hole modifies the geometry of the system by widening the hole. Thus, the size of the holes grows, according to a given law, with the number of pellets already entrapped. The first pellets that fall into holes (those with velocities close to 0, located in the extreme left tail of the distribution) cause the hole size to increase, with the consequence that the critical velocity increases and more pellets are captured because more pellets have a lower speed than v_{cr}. In this way an "avalanche" process is started by the positive feedback resulting from the entrapped pellets, which very quickly brings the proportion of entrapped pellets from 0 to 1 for any speed distribution, provided that there is at least one pellet with a speed lower than the original critical value. The difference between this setting and the preceding one is that now there are indirect interactions between the pellets, and the order parameter behavior depends on the global configuration of the system. Moreover, as the feedback strength depends on the number of entrapped pellets the global number n of pellets present in the box matters since the speed of the process increases with n. The existence of a loop determined by the feedback is reflected in the possibility of plotting the order parameter (the proportion of fallen pellets) as a function of the total number of pellets fallen so far. The same kind of retroaction and avalanche phenomenon has been seen in the case of grammar induction, as was described in Section 11.5.3.

Moving on to relational learning, we showed in Chapter 10 how the change in the probability of finding a matching between a hypothesis and an example influences the quality of the learned knowledge as well as its learnability. Assuming that experiments have uncovered a true phase transition, one has to wonder what the micro and macro levels might be and what kind of interaction could be hypothesized as the basis of the macroscopic behavior. Let us start from the covering test described in Chapter 9. In the covering test, or matching problem (φ, e) between a hypothesis φ and an example e, the macroscopic level corresponds to either an existential or a quantitative problem: the former can be formulated as "Does φ cover e?", whereas the latter can be formulated as "How many models has φ in e?". This is the same distinction as that pointed out by Zweig *et al.* (2010).

Microscopic level in machine learning?

For our purpose we may consider either formulation, as our attention is focused on the less obvious microscopic level. This level must involve the examples. Actually there is more than one way to encode examples as a set of interacting entities, and the one we proposed is just a suggestion. We described in Chapter 9 how an example e comprises a set of m tables, each with N goods; each *good* is a pair of constants, both taken from the same set Λ of cardinality

L. Let us associate with each *good* in each table a node in an m-partite colored multigraph $\boldsymbol{G}_e$ with mN nodes. Given a node (a_i, a_j) from table R_h and a vertex $(a_{i'}, a_{j'})$ from table R_k, the two nodes are connected iff one of the following conditions holds.

- $a_i = a_{i'}$ In this case we connect the nodes with a red edge.
- $a_i = a_{j'}$ In this case we connect the nodes with a green edge.
- $a_j = a_{i'}$ In this case we connect the nodes with a blue edge.
- $a_j = a_{j'}$ In this case we connect the nodes with an orange edge.

Two nodes may then have up to four edges connecting them. Edges can only be established between nodes corresponding to different relations, as each predicate in the domain appears only once in any formula. For this reason the graph is m-partite. The graph captures the internal structure of the example e and defines interactions between the variable pairs. The pattern of interaction determines whether a formula φ covers the example. However, the graph $\boldsymbol{G}_e$ is insufficient for this, because all examples are generated with the same procedure and there is nothing intrinsic in example e that makes it positive or negative with respect to a hypothesis φ. Now comes the second level of interaction: when the formula φ is generated the predicates in it have specified pairs of variables, whose chaining must be matched by the example. Thus there is an interaction between the structure of the formula and the structure of the neutral example that determines whether φ covers e. More precisely, let $\alpha_1(x, y)$ and $\alpha_2(u, v)$ be two predicates in φ, with variables taken from the set $\mathbf{X} = \{x_1, \ldots, x_n\}$.

Let us build a new graph $\boldsymbol{G}_\varphi$, with pairs of variables as nodes. The node (x, y) is connected to node (u, v) with a colored edge using the same criteria as for $\boldsymbol{G}_e$. The graph $\boldsymbol{G}_\varphi$ is m-partite as well, and the node (x, y) associated with the variable pair in α_1 corresponds to the N nodes associated with the pairs occurring in the table α_1 in $\boldsymbol{G}_e$. Knowledge of the formula φ induces a transformation in $\boldsymbol{G}_e$; in fact, given a node (a_i, a_j) in $\boldsymbol{G}_e$ corresponding to the extension of predicate α_1 and another node $(a_{i'}, a_{j'})$ corresponding to the extension of predicate α_2, all edges with a different color from those occurring in $\boldsymbol{G}_\varphi$ between the variable pairs of α_1 and α_2 can be removed, because they will never be part of a model of φ in e. As a consequence, φ will have a model in example e iff $\boldsymbol{G}_e$ contains at least one subgraph isomorphic to $\boldsymbol{G}_\varphi$. The solution of the matching problem is then determined by the global interaction pattern among the goods and by that between the formula and the goods. It is an open question whether these two kinds of interaction generate matching problems whose structure is essentially different for pairs (φ, e) with and without a solution.

Moving to the global level of learning rather than just matching, similar considerations apply. We recall that a learning problem Π consists of a target concept c and examples in two sets, $\mathcal{P}$ (positive) and $\mathcal{N}$ (negative). Possible structural differences between positive and negative examples cannot exist, *per se*, because both sets of examples are generated by an identical procedure. What makes them positive or negative is their interaction with the target concept. In fact, given a set of constants Λ, most examples are located on the horizontal line $L = |\Lambda|$ in the plane (m, L); thus each example is labeled positive or negative either by the location of the target concept itself or when needed by the modification procedures (see Section 10.1.1), which depend upon φ as well.

In the case of learning, we can build two ensembles of graphs, $\mathcal{G}_p$ and $\mathcal{G}_n$, whose elements are associated with the positive and negative examples, respectively. In this case we do not know the target concepts but instead the labels, positive and negative, of the examples are known. Learning consists in extracting commonalities from $\mathcal{G}_p$ and $\mathcal{G}_n$, those from the former to be included in the learned concept, those from the latter to be avoided. This process is again a global one, which exploits the structure of the examples and their links to generalize them. Also, in this case it is an open question whether the internal structures of the ensembles $\mathcal{G}_p$ and $\mathcal{G}_n$, once that they have been labeled by the target concept, are essentially different. As a matter of fact the two ensembles cannot be different, *per se*: they were generated using the same procedure. It is only after the target concept has been defined that the examples are labeled. Thus, it is the interaction with the target concept that may distinguish them. However, examples in $\mathcal{G}_p$ and $\mathcal{G}_n$ can be labelled as positive and negative arbitrarily. In this case it might happen that a concept distinguishing them cannot be found, if the example sets do not have differentiating structures.

Another novel aspect in the paper by Zweig *et al*. (2010) refers to experimental results on the 3-SAT problem. These comprise three issues. The first is that their experimental measurements provide for α_{cr} a value closer to that found by Kirkpatrick and Selman (1994), i.e., 4.15 ± 0.05, than the value found theoretically by Mézard and Zecchina (2002), i.e., 4.267, which is believed to be exact. The second issue consists of a challenge to the phase transition nature of 3-SAT, as they were able to build a simple model that has all the characteristics of the actual k-SAT without invoking any deep restructuring of the solution space. The third issue is the most interesting, from our perspective, as it concerns the number of models. Contrary to what is widely acknowledged, the average number of solutions at the phase transition of a 3-SAT is not 1 and does not drop abruptly to 0 to the right of the critical value α_{cr}; instead, solutions may be found deeper into the UNSAT phase, their average number reaching 1 at the value $\alpha = 5.19$. It is true, however, that this average number is dominated by a few instances with many solutions and a large number of instances with no solution. The

average number of solutions does not show, however, any anomalous behavior at α_{cr}.[3]

This last finding is interesting, not only because it may deeply change the current view of 3-SAT but also because it recalls a phenomenon that occurs in physical phase transitions: a system existing in one phase may go into another by passing the critical point and remaining as it is in an unstable state, which suddenly may change if some perturbation occurs. As a concrete example, let us take water. Very pure water may remain liquid at a temperature well below the freezing point (for instance, at -20 °C at normal pressure) if the temperature is lowered very slowly. The water enters thus an unstable state; it is sufficient to introduce an impurity particle or let the water come into contact with a warmer surface to see it change into ice in almost no time. Owing to the structure of the solution space, it is not unreasonable to think that some instances with $\alpha < \alpha_{cr}$ can be transformed, with small steps, into instances with $\alpha > \alpha_{cr}$ and still be solvable. Clearly, such type of instance must be a minority and the transformation cannot take the original solvable instance too far inside the UNSAT region (actually, $\alpha = 5.19$ is quite close to α_{cr}). This explanation would leave the essence of the 3-SAT phase transition unchanged but it would further complicate the structure of the solution space. Actually, we guess that something similar might happen in learning when positive (negative) examples are slowly transformed into negative (positive) ones. For matching problems (φ, e) that have moved from the YES region into the NO region, near misses (which are negative examples that are almost identical to corresponding positive examples except for a crucial aspect; Winston, 1975; Alphonse and Osmani, 2008a) could be the extremal points that they can reach.

14.2 Do phase transitions occur in practice?

As we have seen in Chapters 3, 4, 9, and 10, generative models for SAT problems, CSPs, and matching and learning problems all have an essentially stochastic nature. It is then a reasonable question to ask whether phenomena such as those investigated in the above-mentioned chapters actually do occur in real life. In order to interpret the emergence of an ensemble phenomenon such as a phase transition, one has to hypothesize that the problems to be solved are extracted from a population of random problems having the same values of the order parameters as those actually considered.

[3] In Chapter 10 we found similar results, as the average number of models that a formula φ has in an example e in the YES region decreases exponentially towards the phase transition but does not show a discontinuity at the boundary.

However, the real world does not seem to us, in general, as random (rather the opposite); evolution has shaped it toward the creation of an environment suitable for life. Thus, given a class of problems and its space of control parameters, the studies that have been performed tell us that the solvable and unsolvable problems are not uniformly distributed and that there are regions where one type of problem dominates the other. Random sampling serves to identify these regions. The fact that in a given region the overwhelming majority of problems is unsolvable does not mean that solvable problems are not present (and vice versa); it means, however, that the latter must be searched for specifically and that we do not have much hope of finding them by just moving around randomly.

There is experimental evidence that a phase transition also occurs in real-world relational problems.

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. Indeed, if the set of training examples is given, the learner is responsible for the generation of the 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 from each other in the way in which they generate hypotheses, i.e., in the heuristics they use. Different heuristics might correspond to phase transitions of a different location and steepness, and the ensemble of matching problems to which they give birth may be more or less similar to the randomly generated set. As we showed in Chapter 10, the space of matching problems has large benches (plateaus), where all heuristics based on coverage are ineffective and the search becomes almost random without additional knowledge. It is no surprise, then, that even in real learning problems a phase transition emerges and that it attracts the search, as in the case of random problem sets (Giordana and Saitta, 2000). In Appendix A two real learning applications, solved by a learner based on an evolutionary search strategy and guided by the minimum description length principle (Anglano *et al.*, 1997, 1998) are analyzed in detail.

However, learning is not an exception. In fact, other authors have previously shown that phase transitions do emerge in real-world problems that are not randomly generated. For instance, Gent and Walsh (1996) 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 also noticed a phase transition in graph coloring problems derived from university exam timetables (Gent *et al.*, 1995); Gomes and Selman (1997) found a phase transition in quasi-group completion.[4]

[4]This term comes from group theory.

Finally, considering again the case of machine learning, it is worth noticing that in the learning problems used in Chapter 10 the examples were not purely random. In fact, negative examples in the YES region and positive examples in the NO region were constructed on purpose. Even so, both the phase transition in the covering test and the consequent blind spot have been detected.

14.3 Blind spot

One of the most surprising effect linked to the occurrence of the phase transition in matching is the presence of a region in the plane (m, L) where learners fail to learn. We called this region, in Chapter 10, a *blind spot*. It includes learning problems that lie in the phase transition region or in the part of the NO region immediately adjacent to it. Considering the findings reported in Chapters 9 and 10, the blind spot can be explained by considering the distribution of models of relational formulas in the (m, L) plane. As can be seen in Figure 10.11(a) the number of models has a peak in the YES region and then decays exponentially toward the phase transition edge. There are three consequences of this distribution.

1. Counting the models in the YES region has an exponential cost for any search algorithm.

2. Finding at least one model for a formula is very easy in the YES region but becomes exponentially hard in the phase transition region. In fact, it requires a very large search tree to be visited, where few or no leaves correspond to a model.

3. The information gain heuristic becomes unreliable, because any systematic difference between the model counts in the positive and negative examples is masked by the large stochastic variance of the model number (see Figure 10.11(b)). For the same reason, any other heuristic based on the model count is expected to fail as well.

The blind spot has quite sharp contours, as might correspond to a threshold phenomenon in the information gain effectiveness; this has an analogue in the coin-tossing problem. In fact, in the case of a binary concept, FOIL's information gain reduces to the computation of the expression

$$IG(h_1, h_2) = t\left[lg_2\left(\frac{P_2}{P_2 + N_2}\right)\right] - lg_2\left(\frac{P_1}{P_1 + N_1}\right),$$

where P_1, P_2, N_1, and N_2 are the number of instances that two hypotheses have on a set of learning events sampled from the world. Thus P_1, P_2, N_1, and N_2 are

inherently stochastic variables and so the information gain itself is a stochastic variable. Therefore, the decision to add a literal to a hypothesis is always done by selecting the maximum value among a set of stochastic variables, as in the case of the coin-tossing problem. When the difference between the number of models (ways of verifying a formula) that a correct concept approximation has in positive and negative instances is small, the probability of distinguishing the correct concept from incorrect approximations will tend to 0.5 and thus the heuristic reduces to a random choice. Even if this point has not yet been experimentally addressed, by referring to Figure 14.1 we can guess that the transition between solvable and unsolvable problems will be quite sharp.

Another important aspect, related to the information gain, is that top-down strategies, used for instance by FOIL (Quinlan and Cameron-Jones, 1993) and SMART+ (Botta and Giordana, 1993), require the number of models that the formula has in every example to be counted. Algorithms like Django only check whether there exists at least one model and stop as soon as this has been found. The process of counting the models, whose number can be very large in the YES region even close to the border with the phase transition region, has a complexity which is lower-bounded by the number of models itself. Thus, the complexity peak is still to be considered as a serious obstacle in scaling up to relational learning problems more complex than those currently addressed.

The information gain heuristic requires the models that a formula has in an example to be counted.

Going into a little more detail, in the YES region every formula has typically plenty of models whereas in the NO region any formula has typically no models. This means that the search trees that must be visited to find a model in the YES region are huge until the phase transition edge is crossed. A formula φ, on the border of the phase transition region, that has been obtained by adding one more literal to a formula ψ may have a search tree of about the same size as that of ψ. The difference is that the leaves of φ's tree are almost all failure nodes while those in the ψ's tree correspond to models. Deciding whether ψ contains at least one model has a low complexity because only a small portion of the tree needs to be visited. At the same time, counting the models of ψ does not help the information gain heuristic because their number is comparably large on all examples. On the contrary, the covering test for φ requires a large part (possibly all) of the tree to be visited; thus regions of the search space where models are few and highly informative for the inductive heuristic are entered by this covering test. As a final observation, in relational learning we notice that, considering the outcomes of the coin-tossing model, we may expect that by increasing the number of positive and negative examples in the learning set it should become easier to detect small differences in the information gain of correct and incorrect concept approximations. Then, the blind spot should shrink. Unfortunately, the tremendous complexity of the learning process in this region has prevented a systematic investigation of this conjecture.

Also in propositional learning, as Baskiotis and Sebag (2004) have shown, there may be regions of the control parameter space where learning tends to fail; specifically, they found such a region for C4.5, guided by the information gain. The considerations for relational learning still hold in this case. In addition, it is plausible to believe that the variance of the information gain estimate increases when a large number of attributes are taken into account in its evaluation. Moreover, we may expect that upon increasing the number of examples, the difference between the information gain of two formulas could be estimated more reliably, so that the blind spot of C4.5 shrinks. Experimental evidence to support this guess would be welcome.

14.4 Number of examples

In the research described in this book, we have not considered the dependence of the results obtained upon the number of training examples. This issue appears to be a relevant one. For instance, as described by Watkin *et al.* (1993), in Ising neural networks a phase transition occurs to perfect learnability in correspondence with a critical value of the number of examples in the learning set (see Chapter 7). Phase transitions with the number of examples as a control parameter have also been detected by Rückert *et al.* (2002) and Alphonse and Osmani (2009).

These findings suggest that relational learning may show a double phase transition, one at the level of matching and one at the level of learning itself. So far, no work reports results on the possible interactions or separation between the two. It would be interesting to investigate their separate contributions to the overall computational complexity. Moreover, an increasing number of examples could constitute a suitable thermodynamic limit for the study of asymptotic behaviors. As we have seen, scaling with the number of variables n does not make much sense in learning, as this value is always limited and actually quite small.

14.5 Machine learning and SAT or CSP solvers

In this book, on the one hand we have presented machine learning as a set of hard search problems, both in the propositional and in the relational framework. On the other hand we have seen how the CSP and SAT fields have progressed, developing powerful search algorithms able to solve problems of a very large size. Particularly impressive are the algorithms for SAT, such as WalkSAT and survey propagation, which are now able to solve problems with millions of clauses and variables.

A natural question is whether, and if so how, such algorithms could be exploited to make progress in machine learning as well, solving problems of a size and complexity that are still intractable today. In fact, we showed in Chapter 10 how a rather naive local search algorithm actually solves many relational problems located in the blind spot, where classical learners fail. Even though an increase in the variable number would render the blind spot unconquerable, we could at least expect to increase the size of the solvable problems by one or two orders of magnitude.

Considering propositional learning, we showed in Section 8.1 how a two-class learning problem can be mapped onto a SAT problem that can be solved by SAT solvers. In turn, a relational learning problem described in a DATALOG language can always be transformed into a propositional learning problem (Kramer *et al.*, 2000). Thus, at least in principle, a relational learning problem can be mapped onto a SAT problem using a two-step transformation. This is surely an interesting approach, worth exploring, which, at least in principle, could solve learning problems that are much more complex than those currently solved by state-of-the-art relational learners. An alternative way, not yet explored, could be to state a relational learning problem as a CSP (which, as a matter of fact, it actually is). Then, either CSP solvers could be used to solve the learning problem or, in turn, the CSP problem could be converted into a SAT problem, as we saw in Chapter 8.

In practice, things are not so simple. The first obstacle is that SAT and CSP exact solvers cannot be used because of the noise which affects all data sets extracted from the real world. So, we must look at MaxSAT and MaxCSP solvers. Nevertheless, in this case also there is an issue to investigate before going further in this direction. MaxSAT and MaxCSP solvers aim at minimizing the number of violated constraints, without considering the nature of these constraints. This strategy may produce solutions that overfit the data without providing useful concept definitions. Actually, this is what happens in the blind spot, where the search heuristics of FOIL and of the other learners fail. As discussed in Chapter 10, in this case the solution is just a description of the learning set which minimizes the number of errors (violated constraints) but does not correctly classify new concept instances because it lacks generalization.

The question is thus how to formulate the problem, or how to provide MaxSAT and MaxCSP solvers with proper heuristics, in order to select good generalizations of the target concept. It is worth noting that a major stream in machine learning, that based on the kernel approach (Shawe-Taylor and Cristianini, 2004), formulates a learning problem as a task of minimizing the number of violated constraints. The difference from the approach we are proposing here is that kernels are actually continuous functions and the task is solved with methods developed in linear algebra.

14.6 Relational learning and complex networks

We have seen that every single example used by relational learning can be represented as a colored graph (network) of items. A subset of these examples, i.e., the positive examples in the YES region and the negative examples in the NO region, are actually constructed as random graphs and thus they must exhibit the properties discussed in Section 4.2. More specifically, we should expect a phase transition in the connectivity in correspondence with the critical value $z_{cr} = 1$ of the degree z of the graph. However, the average degree z of the examples randomly generated to investigate relational learning is approximately

$$z = 2(m-1)\left(\frac{N}{L^2}\right),$$

which is usually much larger than the critical value $z_{cr} = 1$. This means that most of the time all considered examples are strongly connected. Nevertheless, for N very small ($N < L^2/2(m-1)$), the examples will reduce to a set of small independent substructures and the learning problem will be located very far from the phase transition, easy to solve, and not interesting for our purpose.

Let us consider the strongly connected examples that we actually generated. In this case, the models of a formula φ in an example e correspond to subgraphs in the graph $\boldsymbol{G}_e$ of the example. Finding a solution to the problem of the covering test reduces to finding a subgraph in $\boldsymbol{G}_e$ that is isomorphic to $\boldsymbol{G}_\varphi$. This is the viewpoint taken in the development of algorithms like MCS (Scheffer *et al.*, 1996).

In any case, looking at structured learning examples as graphs is suggestive and opens other perspectives for future research. In the first place it establishes an explicit link between relational learning and investigations of complex networks. Discovering a "community", like, for instance, Internet users, in a graph is equivalent to describing a subgraph satisfying a set of preassigned conditions. Thus it can be modeled as a relational learning problem. However, a method for discovering communities in social networks should be easy to adapt to the problem of verifying a logical formula.

Another intriguing point to investigate is the impact of the graph structure on the performances of relational learning algorithms. The blind spot for FOIL was found by considering examples whose background structure is an almost homogeneous random graph. Singularities constructed by the procedure **ChangeToPositive** described in Section 10.1.1, correspond to a single (non-random) subgraph forcibly inserted into a random graph. Then, solving a learning problem in the blind spot reduces to finding and characterizing a non-random subgraph in a large set of random subgraphs.

Thus, in learning, two types of graphs are involved: the first is the constraint graph and the second is the graph associated with each example. Both have the underlying structure of a random graph, such as those introduced in Definitions 3.6 and 3.7. It would be interesting to investigate what would happen if the generation procedures considered here were replaced by a *small-world* or *scale-free* graph generator. With very few exceptions (see for instance Walsh, 1999), this issue has not been approached so far.

14.7 Relational machine learning perspective

The most appealing properties of learning in first-order logic is the comprehensibility of the acquired knowledge and the ability to learn in complex domains where objects cannot be simply represented by $\langle attribute, value \rangle$ pairs. However, according to the findings reported in this book, relational learning also has strong limitations, especially relating to the number of variables; in fact, we cannot expect to scale up to clauses with more than four or five variables without incurring a prohibitive computational complexity. Therefore relational learning, as we now know it, is probably restricted to dealing with data and concepts of limited size, however interesting. Moreover, as the phase transition region acts as an attractor, the quality of the learned knowledge is also affected because only hypotheses in that region may actually be learned. These facts are likely also to affect statistical relational learning, or at least those approaches that learn first the network structure and then the parameters.

Coming back to the statistical physics perspective and assuming that the experimentally uncovered phase transition is a "true" one, several questions arise that need to be answered. We saw in Chapter 2 that, according to a modern view, a phase transition occurs when the partition function of the system under analysis has a singularity. If we want to transfer results from statistical physics to relational learning, we need to identify what the corresponding "partition function" might be. This function is likely to be linked to the structure of the examples and to their interactions, both within themselves (the connections between tuples of constants in one example) and among them (commonalities to be discovered). In this way, we could be in the same situation as that for Ising neural networks and for SAT problems, where a precise definition of the partition function can be obtained and methods from statistical physics can be applied.

Moreover, given the strict relation between matching and CSP, the solution space of matching also might show an analogous clustering structure. Investigating this aspect has the potential of suggesting more effective learning strategies, in the same way as it suggested the survey propagation algorithm for SAT. Concerning the search algorithms we have seen that, by exploiting the powerful

heuristics developed in the CSP field, the complexity of the covering test can be at least partly reduced. This opens up the possibility of handling hypotheses that are much more complex that those currently generated by relational learners. We may wonder whether an analogous step ahead in the area of the heuristics for guiding the inductive search is likely. The stochastic algorithm presented in Chapter 10 proved to be capable of solving many problems not solved by FOIL, while its complexity was comparable. More specifically, we showed that by combining stochastic search with local deterministic search it is possible to learn approximated concept descriptions where no known classical algorithm is successful. Even if the algorithm is used under the stringent assumption that a conjunctive concept description exists, it is not difficult to extend it to cope with more general concept descriptions. For instance, disjunctive descriptions can be learned by integrating the algorithm $\mathbf{T}^4$ (Section 10.4.2) with the type of set-covering algorithm found in most relational learners (Quinlan and Cameron-Jones, 1993; Botta and Giordana, 1993). As a matter of fact, the approach described, even if still in a very elementary version, goes along the lines of CSP heuristics such as local search and tabu search.[5] Thus we expect that more powerful relational learners can be designed and implemented.

Finally, in machine learning in general it is well known that the selection of the learning set $\mathcal{S}_L$ can deeply affect the robustness of the learned knowledge, especially for unstable learners such as decision tree learners. As we have seen in previous chapters, $\mathcal{S}_L$ acts as the quenched disorder of the learning system. An interesting question is whether the generalization error is a self-averaging quantity, such that, for increasing cardinality of $\mathcal{S}_L$, it becomes independent of the learning set, so that only its mean value need be computed, not its whole distribution. This question has received a positive answer in the case of neural networks, as we saw in Chapter 7.

A point that seems a real obstacle to the scaling up of relational learning is the number of variables. Even under the simplistic assumption that all predicates are relevant to the concept description, the task looks hard for many concepts requiring at least four variables. On increasing the number of variables, the complexity rises exponentially. Given the presence of irrelevant predicates, the analysis we performed still holds but the density of subformulas of the target concept close to the phase transition becomes even smaller, and so the difficulty increases further. This limit on the number of variables means, for instance, that we cannot learn descriptions of scenarios with more than four objects in them.

After pointing out the difficulties of the task, we will mention possible ways to mitigating the negative aspects of the relational learning task. Beyond the potential improvements in the search algorithms mentioned above, there appear to

[5] A tabu search has a memory of past failures, kept to avoid repeating them.

be three ways to scale up relational learning, from the point of view of reducing its computational needs and improving the quality of the results, which we now discuss.

Propositionalization

In relational machine learning much effort has been devoted to *propositionalization*, i.e., the process of transforming a relational learning problem into a propositional one, with the aim of exploiting the efficient propositional learning algorithms that are available (Lavrač and Džeroski, 1994; Alphonse and Rouveirol, 2000; Bournaud *et al.*, 2003; Mauro *et al.*, 2010). Reviews have been presented by Kramer *et al.* (2000), Krogel *et al.* (2003), and Karunaratne and Boström (2009).

The idea is to transform a multirelational data set containing structured examples into a propositional one with grounded $\langle attribute, value \rangle$ features describing the structural properties of the examples. The propositional data set contains a single table, with one row per example. Propositionalization has proved to be an effective method in relational learning, and several algorithms implementing the process are available in the literature.

The process has, however, two potentially negative aspects. The first is that the size of the target table may grow exponentially in the size of the original problem, so that the advantage of using a fast propositional algorithm is reduced by the very size of the data. The second is that the learned knowledge may only be an approximation to what could have been learned in the first-order logic setting. In fact, by aggregating and summarizing the original data some information may be lost.

Problems solved by current relational learners are easy to transform into the propositional framework.

Nevertheless, propositionalization may be a viable way of approaching the problem. In this case it would be interesting to investigate where the phase transition in the matching of the original problem lies. Furthermore, an accurate investigation of the issue may suggest more effective ways of transforming the original data. It could be more rewarding to invest in techniques for translating a relational problem into a propositional one than in new algorithms for learning directly in the relational framework.

***A priori* knowledge**

Another way of scaling up to more complex problems could be to exploit domain knowledge to guide the search. As an example, Alphonse and Osmani (2009) showed that by providing the right knowledge, in the form of "near-miss" negative examples, it is possible to solve all the problems in the dataset we used for the experiments described in previous chapters. This agrees with the principle on which the stochastic approach we proposed is based. If we know a *good*

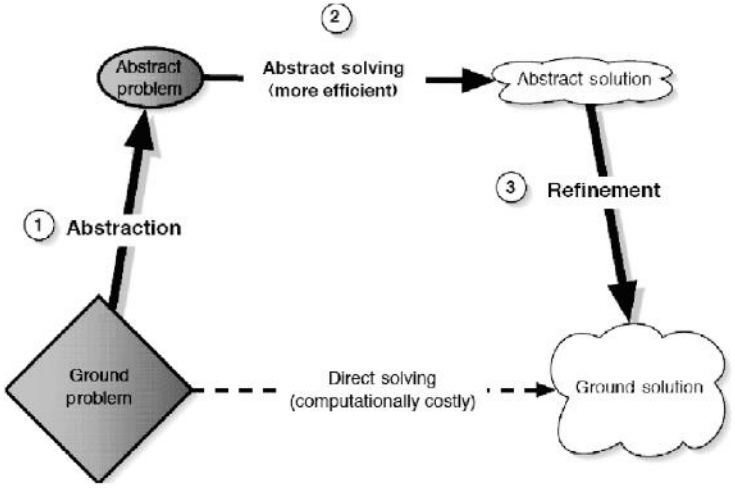


Figure 14.3 Abstraction process for problem solving. In learning, the ground problem consists of the representation spaces of examples and hypotheses. These spaces are then abstracted, obtaining some "simpler" spaces where learning may take place. After learning, the acquired knowledge may be re-transformed into the original space even though in some cases this step may not be necessary.

educated guess to start from, we would not need stochastic sampling on the hypothesis space.

In a random domain like the one we explored, no domain knowledge is possible. Nevertheless, in many real-world applications this knowledge is available and can be exploited to prune the search in the hypothesis space. Of course, coding the necessary knowledge and developing algorithms for using it is costly but may be a winning strategy in the long run.

Considering the many complex problems successfully solved in fields such as signal interpretation and molecular biology, we can see that there has been substantial progress in discovering the underlying structure of a problem when learning techniques, typically in a propositional framework, have been exploited. In principle one could think of approaching the same kind of problem in a relational framework that exploits the expressive power of first-order-logic languages. An interactive approach, where human experts and machines cooperate to learn in complex domains might be the solution.

Knowledge-based approaches may be a valid option.

Abstraction

Another way to try to reduce the negative effects of a phase transition in learning problems is to use *abstraction*. Abstraction is a pervasive activity in human

perception, conceptualization, and reasoning. It is generally associated with a transformation of a problem representation that allows a solution to be found more easily, i.e., with *reduced computational effort*. The process is represented in Figure 14.3. Abstraction seems a very promising way out of the limitations of relational learning. Indeed, previous works have already shown that, when a "good" abstraction can be found, relational learning may be achieved with the same quality of acquired knowledge and with a strong reduction in computational cost (Saitta and Zucker, 2000). However, the type of abstraction has to be chosen carefully, because it always entails an information reduction. This loss of information may be critical, and the learning problem might not be solvable in the abstract space.

Leaving aside all considerations regarding the choice of a good abstraction (which are outside our current scope), we just provide a hint of how it may work. Among various abstraction operators (Saitta and Zucker, 2000, 2001, 2009) let us consider the *term construction* operator, which inputs a description of the parts of an object, and of how they are related, and outputs a single composite description. For instance, the whole object "bicycle" can be obtained from two "wheels", one "handlebar", two "pedals", and so on. As long as we do not need to distinguish its parts, the whole object may be used for learning. The big advantage of an abstract space where only whole bicycles exist is that each bicycle can be associated with a single variable whereas in the concrete space many variables were needed to describe it. As we have seen that the number of variables is the strongest limiting factor in relational learning, this type of abstraction moves the learning problem into a space where a phase transition still exists but is located in a region of the control parameters where the complexity is lower.

Another approach exploiting abstraction to move away from the phase transition in CSP is described by Schrag and Miranker (1996). They used domain abstraction to allow some subset of constants appearing in the relations to collapse to single constants. They showed that the transformation loosens the constraints so that this type of abstraction is effective when both the original and the abstract CSP are unsolvable. Through this type of abstraction, when effective, a sensible reduction in computational cost is obtained.

The explicit use of abstraction techniques for the same problem class was proposed by Caseau (1991) and Ellman (1993).

Beyond the ideas suggested in this section, other approaches may be considered, in order to improve relational learning with respect to both quality and computational cost. Up to the present time only simple learning applications have been described, so that the negative effects of the phase transition in matching have been limited. For the future, either we will not need more complex learning cases or we will need to devise means to cope with them.

Appendix A Phase transitions detected in two real cases

In Chapter 9 we claimed that there is experimental evidence that in real-world applications also, where examples are not randomly generated, discriminant hypotheses found by relational learners lie on the phase transition edge. In order to support this claim, we discuss here the findings presented by Giordana and Saitta (2000) concerning two-real world applications. The first is a popular benchmark known as the mutagenesis dataset (Srinivasan *et al.*, 1995), while the second is an application to mechanical troubleshooting in a chemical plant (Giordana *et al.*, 1993). In both cases the learning problems were solved using G-Net, the relational learner based on evolutionary search described in Chapter 6 (Anglano *et al.*, 1997, 1998).

It is worth noticing that datasets suitable for relational learning and available in public repositories are few and, in general, rather simple. In fact, the concept descriptions that have been learned from them contain few literals only and, mostly, two or three chained variables. The datasets that we present in this appendix are among the most complex approached with machine learning: for both, descriptions containing up to four variables and up to six binary relations have been discovered. For the sake of reference, Figure A.1 gives the same graph as Figure 9.9(a) but for $n = 4$. A phase transition is evident, but the expected complexity in the mushy region is much lower than that in Figure 9.9(a).

Comparing Figure A.1 ($n = 4$) with Figure 9.9(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 and Hogg, 1994). Moreover, a 50-fold increase in the complexity is observed in correspondence with a 2.5-fold increase in the number of variables.

A.1 Mutagenesis dataset

The *mutagenesis* dataset has been widely used in the machine learning community as a benchmark for testing induction algorithms in first-order logic. The task

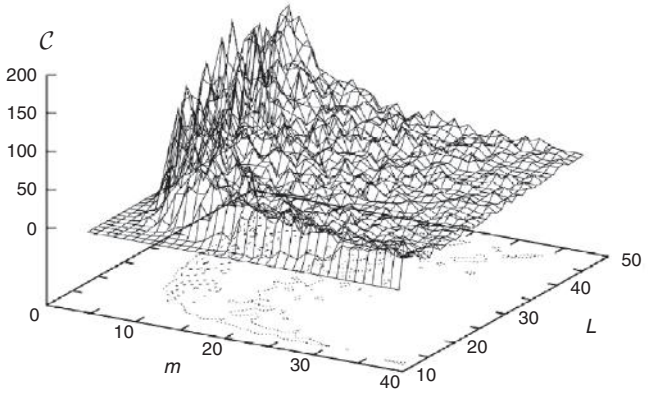


Figure A.1 Complexity in the (m, L) plane for randomly generated matching problems with $n = 4$ and $N = 100$.

is to learn to predict the mutagenicity in nitroaromatic chemical compounds on the basis of their structure (Srinivasan *et al.*, 1995). The goal of our analysis was to investigate where the classification rules learned by an inductive program lie in the plane (m, L), with respect to the mushy region.

The mutagenesis dataset[1] 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 characterized by an attribute vector reporting the atom type, the atomic number, and the electrical charge, plus a set of relations describing atomic links and substructures of the molecule such as aromatic rings. 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. Extensive experimentation with different sets of attributes was reported by Srinivasan *et al.* (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 analysis with the method used in Chapter 9, which was limited to binary constraints.[2] However, the problem can be reformulated using only unary and binary predicates, as was done by Anglano *et al.* (1998). Every molecule is considered as

[1] The dataset used here is a "regression friendly" one: it includes those examples that can be modeled with a good approximation by linear regression.

[2] A discussion on the relations between binary and non-binary CSPs was provided by Bacchus and van Beek (1998).

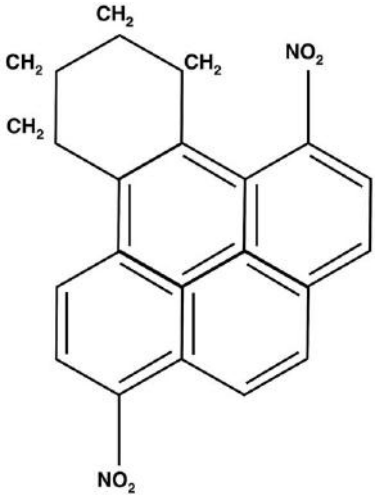


Figure A.2 Example of a nitroaromatic molecule structure in the mutagenesis dataset. Each atom is denoted by a constant, and each link defines a binary relation between two atoms.

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 over atoms and K denotes a set of constants that are to be learned by an induction algorithm (Giordana *et al.*, 1997). In Figure A.2 an example is shown.

A set of experiments was performed for each of two different hypothesis description languages, $\mathcal{L}_1$ and $\mathcal{L}_2$. The language $\mathcal{L}_1$ was analogous to that used by other authors in the past (Sebag and 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 or otherwise of a link between two atoms. Moreover, the constraint $x < y$ was 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 atom. The algorithm G-Net was programmed 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

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

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

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

$$\varphi_4 : anm(x_1,[195,3,27,38,40,29,92]) \wedge \neg type(x_1,[H]) \wedge \neg chrg(x_1,[-0.2])$$
$$\neg anm(x_3,[40]) \wedge anm(x_4,[195,22,27,38,40,29,92]) \wedge \neg type(x_4,[H,N])$$
$$(x_1 < x_2) \wedge \neg bound(x_1,x_2) \wedge (x_1 < x_3) \wedge (x_1 < x_4) \wedge (x_2 < x_3) \wedge$$
$$(x_2 < x_4) \wedge bound(x_3,x_4) \wedge (x_3 < x_4) \rightarrow mutagenic.$$

Figure A.3 The solution Φ learned by G-Net using the language $\mathcal{L}_1$; Φ correctly classifies 94.1% of the dataset.[4]

covering from 90% to 95% of the examples depending on the control parameter setting.[3]

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 shown in Figure A.3, which are expressed in the language $\mathcal{L}_1$, and $\Psi = \{\psi_1, \psi_2, \psi_3\}$ consisting of the three clauses shown in Figure A.4, which are expressed in the language $\mathcal{L}_2$. The same analysis was performed on several other solutions generated by G-Net, and qualitatively equivalent results were obtained.

All rules in the solutions Φ and Ψ were analyzed according to the following procedure. For each rule $\varphi_i \in \Phi$ or $\psi_i \in \Psi$, the two parameters p_2 and $\hat{p}_{2,cr}$ were computed for every example in the dataset. The reasons for using p_2 were that 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. Thus, theoretical results from the literature (Prosser, 1996) can be used directly.

[3] In these experiments the whole dataset was used, because here we were interested not in evaluating the predictive power of the learned knowledge, but in the impact of the matching complexity on the learning process.

[4] In the φ_i the symbol $\neg$ denotes, as before, the negation of a variable.

$$
\begin{aligned}
&\psi_1 : \ first-atom(x_1) \wedge logp(x_1, [0.0 \div 7.0]) \wedge \neg lumo(x_1, [-1.0]) \wedge \\
&\quad \neg logp(x_2, [1.5, 7.0]) \wedge \neg lumo(x_2, [-1.25]) \wedge \neg logp(x_3, [0.5, 1.0, 6.5]) \wedge \\
&\quad \neg lumo(x_3, [-4.0 \div -1.0]) \wedge \neg logp(x_4, [2.5, 3.0]) \wedge (x_1 < x_2) \wedge (x_1 < x_3) \wedge \\
&\quad (x_1 < x_4) \wedge (x_2 < x_3) \wedge (x_2 < x_4) \wedge (x_3 < x_4) \ \rightarrow \ mutagenic, \\
&\\
&\psi_2 : \ first-atom(x_1) \wedge logp(x_1, [0.0 \div 7.0]) \wedge \neg lumo(x_1, [-1.0]) \wedge \\
&\quad \neg logp(x_2, [1.5]) \wedge \neg lumo(x_2, [-1.25]) \wedge \neg logp(x_3, [0.5]) \wedge \\
&\quad lumo(x_3, [-1.5, -0.75]) \wedge \neg logp(x_4, [2.5]) \wedge \neg lumo(x_4, [-1.75]) \wedge (x_1 < x_2) \\
&\quad \wedge (x_1 < x_3) \wedge (x_1 < x_4) \wedge (x_2 < x_3) \wedge (x_2 < x_4) \wedge (x_3 < x_4) \ \rightarrow \\
&\quad mutagenic, \\
&\psi_3 : \ first-atom(x_1) \wedge \neg lumo(x_1, [-1.0]) \wedge \neg logp(x_2, 2.0]) \wedge \\
&\quad anm(x_3, [195, 22, 3, 27, 38, 40, 29, 92]) \wedge \neg chrg(x_3, [-0.20)]) \wedge \\
&\quad \neg anm(x_4, [22]) \wedge type(x_4, [C, O, F]) \wedge \neg chrg(x_4, [-0.4, 0.0]) \wedge \\
&\quad (x_1 < x_2) \wedge (x_1 < x_3) \wedge (x_1 < x_4) \wedge (x_2 < x_3) \wedge \\
&\quad (x_2 < x_4) \wedge (x_3 < x_4) \ \rightarrow \ mutagenic.
\end{aligned}
$$

Figure A.4 The solution Ψ learned by G-Net using the language $\mathcal{L}_2$; Ψ correctly classifies 90.7% of the dataset.

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

$$\gamma(x_1, x_2) = \alpha_1(x_1) \wedge \alpha_2(x_2) \wedge \beta(x_1, x_2). \tag{A.1}$$

Each subformula γ was considered as a single constraint. The unary predicates occur in each subformula containing the same variable as an argument; 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 were always obtained. Thus, $p_1 = 1$ for every clause, whereas the parameter $\hat{p}_{2,cr}$ 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 was $L_{min} = 18$, the maximum was $L_{Max} = 40$, and the average was $L_{avg} = 26.7$.

Using the expression (9.21) we obtain, considering all formulas:

$$\hat{p}_{2,cr} = 1 - L^{-4/6.7} \tag{A.2}$$

The parameter p_2 also depends upon the formula φ and upon the universe U represented by an example e; in order to stress this dependency, we use the notation $p_2(\varphi, e)$. More specifically, p_2 was computed according to the expression

$$p_2(\varphi, e) = \frac{1}{6} \sum_{j=1}^{6} p_2(\gamma_j, e) = 1 - \frac{1}{6L^2} \sum_{j=1}^{6} N_j. \tag{A.3}$$

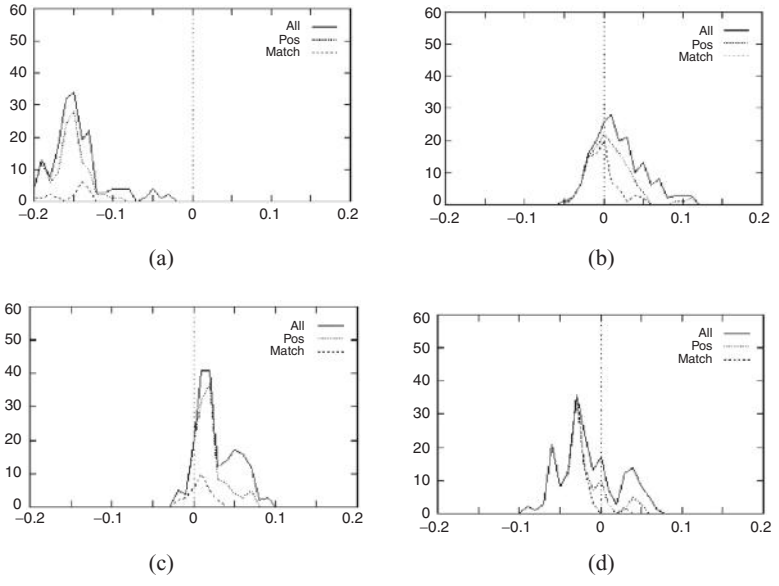


Figure A.5 Distribution of the variable $p_2 - \hat{p}_{2,cr}$ for the mutagenesis dataset, for (a) φ_1, (b) φ_2, (c) φ_3, (d) φ_4 vs. the number of examples (all, positive, and those matched by the formula).

In (A.3), γ_j is a binary subformula 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 $p_2 - \hat{p}_{2,cr}$ 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 shown in Figure A.5. If the matching problem corresponding to a (φ, e) pair is exactly on the phase transition then the value $p_2 - \hat{p}_{2,cr}$ is zero. Notice that the mushy region is quite large for $n = 4$, as we can see from Figure A.1; moreover, as neither L nor N are constant across relations and examples, the broadening of the mushy region is enhanced. Figure A.5 clearly shows that, for the 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 lie mostly in the YES region.

The same analysis was performed for the solution Ψ, and the results are shown in Figure A.6. The Solution Ψ shows a different behavior from Φ. In fact, the rules ψ_1 and ψ_2 exhibit three separate peaks: one to the left, one inside, and one to the right of the mushy region, respectively. Moreover, in each case the peaks corresponding to the examples satisfying the rule practically coincide with the left-hand peak. A different behavior is exhibited by rule ψ_3, which shows only two peaks, the first near the critical point $\hat{p}_{2,cr}$, and the second clearly to the

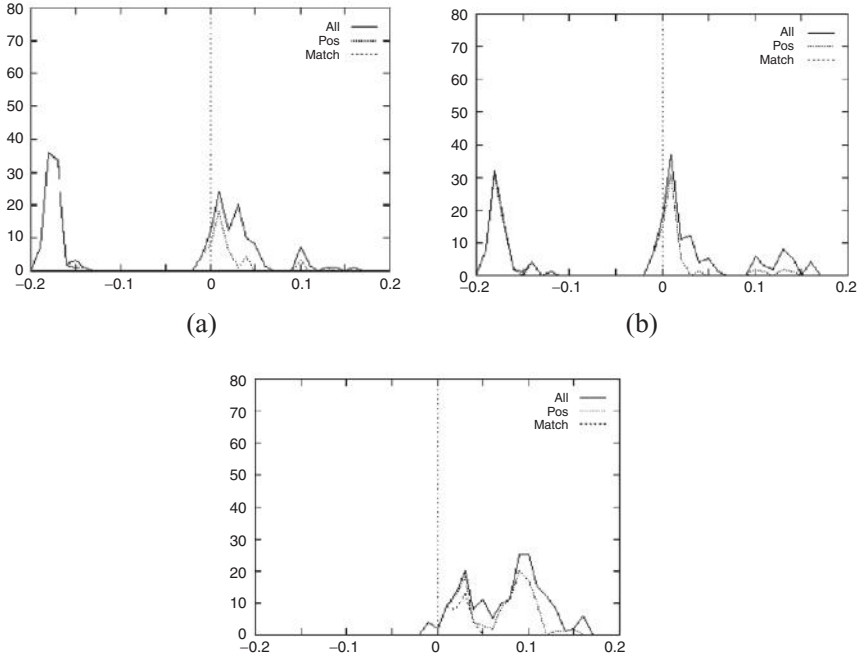


Figure A.6 Distribution of the variable $p_2 - \hat{p}_{2,cr}$ for the mutagenesis dataset, for (a) ψ_1, (b) ψ_2, and (c) ψ_3, vs. the number of examples (all, positive, and those matched by the formula).

right of the mushy region. This situation is confirmed by the presence of both positive and negative instances in the peaks.

From Figures A.5(a)–(d) we predict that formula φ_1 should be easy to match for all the examples, whereas matching φ_2 is likely to involve a high computational cost because most examples lie 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 A.1 the measured complexities for matching the formulas on the whole dataset are reported. As can be seen the theory prediction for all the formulas is substantially verified, except for φ_1, for which both the location of the peak in Figure A.5(a) and the complexity in Table A.1 appear to be in error. Looking more closely at formula φ_1 in Figure A.3, we suggest the following explanation. On the one hand, the formula φ_1 actually contains only two "meaningful" variables, namely x_3 and x_4, in the predicate $bound(x_3, x_4)$; thus, both n and m are overestimated and so the value $\hat{p}_{2,cr}$ is actually a little larger than that in the figure. On the other hand, N is computed as the average of all the relations involved in the formula,

Table A.1 Average complexities for matching the clauses in Φ and Ψ to the examples of the dataset

	Φ				Ψ		
	φ_1	φ_2	φ_3	φ_4	ψ_1	ψ_2	ψ_3
Avg	26 215.10	5168.06	1249.04	1496.85	1.33	1.43	7.06
Avg_{pos}	22.46	207.74	23.89	1249.86	2.00	2.00	2.35
Avg_{neg}	30 418.86	8609.00	1463.44	1789.79	1.00	1.00	8.33

Table A.2 Classification rates obtained by setting a threshold between the peaks corresponding to low and high p_2 values, respectively, for the three formulas ψ_1, ψ_2, and ψ_3. The values within parentheses 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)

so that the extension of $(x_1 < x_2)$, which is much larger than the other, means that p_2 appears much smaller than it must be. The consequence is an apparent shift toward the left with respect to the phase transition. The second aspect to be explained, namely the abnormally high complexity of φ_1 seen in Table A.1, is also related to spurious joins of the intermediate tables corresponding to x_1 and x_2, which are pruned only later. This effect would not have appeared if a dynamic variable ordering had been exploited during matching. A set of focused experiments in which φ_1 was reduced to the subformula containing only x_3 and x_4, confirmed both explanations. Of the seven formulas in Φ and Ψ, φ_1 is the only formula in which only two variables are effective. It is sufficient that three among the four variables are chained by the predicate *bound*, which is much more constraining than the predicate $<$, for the anomaly to disappear.

An interesting observation can be made for Figure A.6(a)–(c): the positive and negative examples could be discriminated almost without performing the matching but simply by setting a threshold on p_2: by considering as "positive" the examples on the left and as "negative" those on the right of the threshold, the classification reported in Table A.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 complexities in Table A.1 confirm this assertion. The above property can be exploited to reduce the amount of matching to be done during learning and knowledge use. By estimating the distributions of p_2 values for the positive and negative training examples, a "best" threshold (or, preferably, a "best margin") can be learned.

Moreover, by looking at the syntactic structure of the clauses in Ψ, (see Figure A.4), we notice that most literals occurring in them involve 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 their structural aspect, ψ_1 and ψ_2 are easily translated into propositional assertions. The rule ψ_3 shows a different structure, since it also contains 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 the solution Φ.

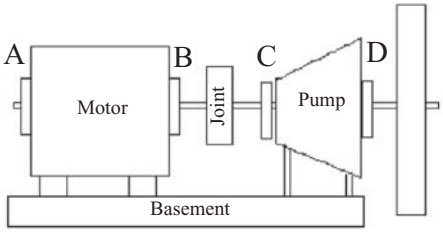
A.2 Mechanical troubleshooting datasets

The second real-world case study that we discuss here is a problem approached by two of the authors 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. The knowledge base learned by the system ENIGMA (Giordana *et al.*, 1993) has been used for many years by the company.

The basis for the troubleshooting was mechanalysis, a methodology that exploits mechanical vibrations and requires considerable expertise for its application. The diagnosed apparatus, ranging from small motor pumps to 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 measured on the supports of the machine components. Each mechanalysis constitutes an example. The data, arranged into groups, correspond to the machine's supports: each group contains the frequency and velocity of the harmonic components of the vibration for three spatial directions, as shown in Figure A.7.

The troubleshooting task consists of discriminating between six classes (one "normal" and five for the types of fault). G-Net found 13 conjunctive formulas, distributed over the six classes,[5] each having at most four variables. One of these

[5] In the real-world application the system ENIGMA was used (Giordana *et al.*, 1993), but here we have re-analyzed the dataset using the new system G-Net. In fact, the knowledge base used in the field was obtained by an integration of similarity- and explanation-based learning, and



(a)

		Total vibration		Fourier analysis				
		Amplitude	Speed	ω	v		ω	v
Support	Direction	(μm)	(mm/s)	(CPM)	(mm/s)	...	(CPM)	(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 A.7 Structure of a mechanalysis table corresponding to a single example. (a) Scheme of a motor pump. The vibrations of 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 those given under the heading "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.

formulas is the following:

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

The meaning of the predicates in φ is not important here and can be found in Giordana *et al.* (1993). The relevant aspect is the syntactic structure of φ. In Figures A.8 and A.9 the results of the same analysis as that performed on

was structured with chains of disjunctive rules instead of flat ones. In the cited paper a complete description of the application may be found.

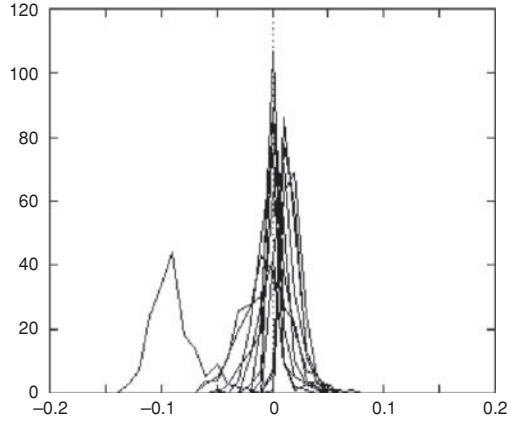


Figure A.8 Distribution of the variable $p_2 - \hat{p}_{2,cr}$ for the matching problems obtained by pairing each of the 13 formulas (disjuncts) in the solution with all the examples in the dataset. Each graph corresponds to one of the 13 formulas.

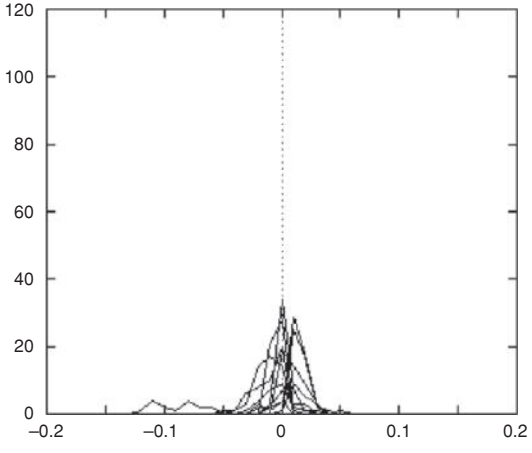


Figure A.9 Distribution of the variable $p_2 - \hat{p}_{2,cr}$ obtained by matching each disjunct corresponding to a given class with the positive examples of the same class that are covered by it. Hence, all the considered problems are solvable.

the mutagenesis dataset are shown. More specifically, Figure A.8 reports the distribution of the variable $p_2 - \hat{p}_{2,cr}$ for the matching problems obtained by pairing each of the 13 formulas with all the examples in the dataset (164 examples), giving a total of 2132 matching problems. In Figure A.9, however, only matching problems obtained by pairing each formula with the positive examples of its class are considered.

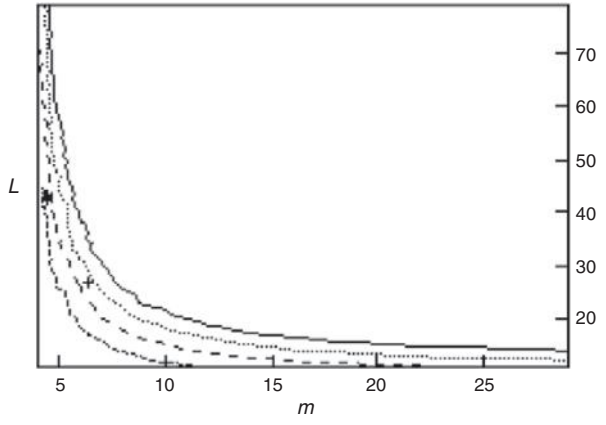


Figure A.10 Location of the line $P_{sol} = 0.5$ for $N = 50, 80, 100, 130$ and $n = 4$ variables. The symbols $+$ and * (this lies about halfway up the vertical axis) locate the positions in the plane (m, L) of the "average" matching problems found in the mutagenesis and mechanical troubleshooting datasets, respectively.

As we can see from Figure A.8, most problems lie inside the mushy region, except for one formula. A closer analysis of this formula showed that, in contrast with the case of Figure A.1(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 that we have considered, the cardinality N of the relations corresponding to the basic predicates was not constant, as is assumed in model RL. Thus, we considered the model prediction for a range of N values corresponding to the actual cardinalities occurring in the two datasets. The plot in Figure A.10 is analogous to that in Figure 9.8(b) but for $n = 4$ variables. Again, N was set to 50, 80, 100, and 130, respectively.

In Figure A.10 we have indicated the "average" solutions found by G-Net (the average was over all pairs (φ, e), in the plane (m, L)). As can be seen from the figure, these solutions are located on the respective phase transition curves.

Appendix B An intriguing idea

It is easy to produce a probability function that exhibits a very steep transition between the values 0 and 1. Take for instance a binary tree corresponding to the exploration graph of a two-player game with a constant branching factor b. Each node in the tree represents a position, and each edge a possible move for a player from one position to a next position. Some games do indeed offer only exactly b possibilities at each time to the current player.

Suppose further, as it is usually the case, that the computer whose turn it is to play does not have sufficient time or memory space to explore the whole tree of possibilities. Then, the standard approach is for the computer to develop the tree to a given depth, say 10, and then to evaluate the merit of each position and to carry up these estimations through the celebrated min–max procedure. If a node represents the computer's turn to play, the maximum value of the nodes below is returned and passed above, otherwise the minimal value is passed above.

One question is then how to compute the probability of a "win" at the root of the tree given the probability that a leaf node is a win. By computing the probability that a node is a win, given the probability of a win of its direct successors and its own nature (either a Max node or a Min node), it is straightforward to compute the probability at a Max node at depth $d - n$ from the probability of a Min node at depth $d - (n - 1)$, one move ahead: thus

$$P_n = 1 - (1 - P_{n-1}^b)^b, \tag{B.1}$$

where b is the branching factor and P_n is the probability of a win when n is the number of moves to be played by Max until a leaf node (where $n = 0$) is reached. This is a recursive function (called in this case the *logistic* function), from which the probability of a win at the root can be calculated given the probability of a win at a leaf node, $P_d = f(P_0)$. The function $f(x) = 1 - (1 - x^b)^b$ intersects the line $y = x$ at three points. Two of them, at $x = 0$ and $x = 1$, are stable points and the third, $x = \xi_b$, is unstable. If P_{n-1} is equal to any of these three values then we have $P_n = P_{n-1}$. If, however, $P_{n-1} < \xi_b$ then $P_n < P_{n-1}$ and therefore $P_{n+1} < P_n < P_{n-1}$, and so on, whereas if $P_{n-1} > \xi_b$ then $P_n > P_{n-1}$ and therefore $P_{n+1} > P_n > P_{n-1}$, and so on (see Pearl, 1984).

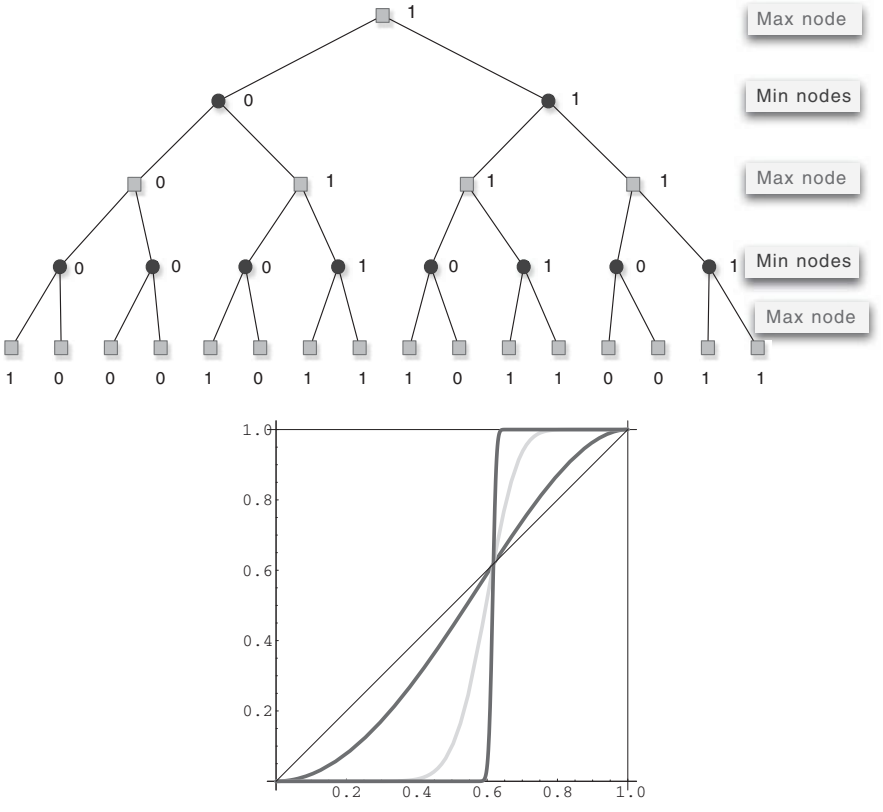


Figure B.1 (Lower panel) The probability (see (B.1)) of a win at the root node of a max–min game with branching factor $b = 2$ (upper panel) when the depth of the game increases from $n = 1$ to $n = 5$ and $n = 10$.

As a consequence, the greater the value of the depth d, the steeper the function f (see Figure B.1). In the limit, we have

$$\lim_{d\to\infty} P_d(P_0) = \begin{cases} 1 & \text{if } P_0 > \xi_b \\ 0 & \text{if } P_0 < \xi_b \end{cases} \tag{B.2}$$

The same analysis can be carried out to compute the probability of acceptance of a string when an automaton has a self-similar structure.

Suppose that we have a "circuit" of the form shown of Figure B.2 (upper panel). We will call such a circuit a *series circuit* by analogy with the electrical case. Each edge in the circuit acts as a filter. Either the current letter in the incoming string matches the letter specified by the edge and the remaining string goes

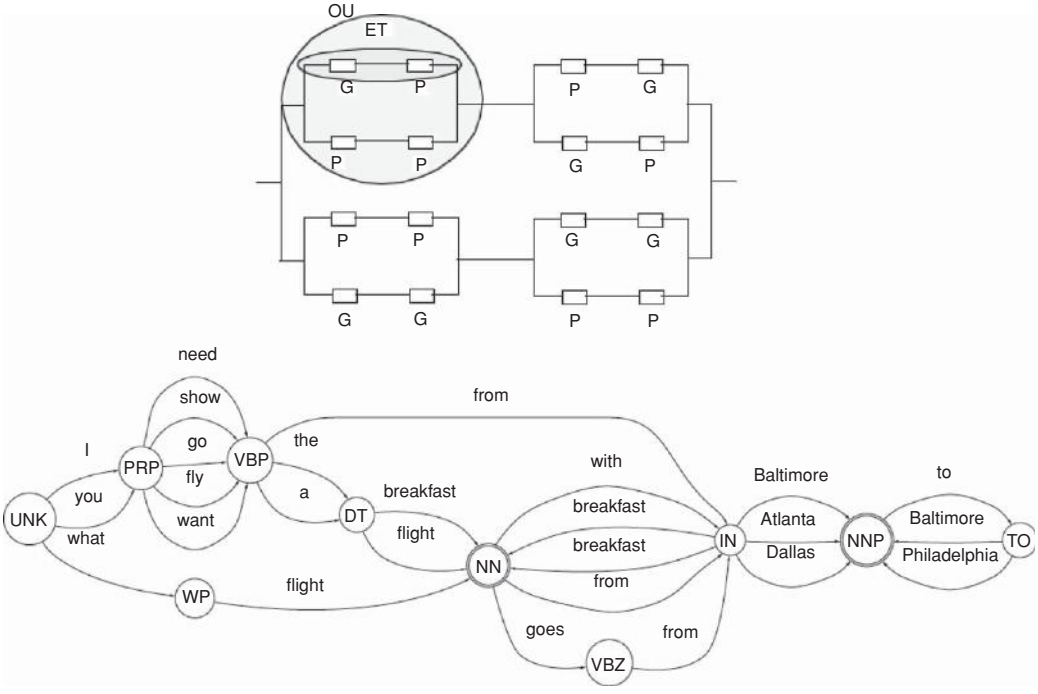


Figure B.2 (Upper panel) A self-similar series–parallel circuit. The top left ellipse encloses a series circuit. The probability that a string passes through the circuit is $P_{n-1} = Q_n^2$ if the probability that the string passes each test is Q_n. The larger ellipse encloses an elementary series–parallel circuit with two branches. The probability that a string passes it is $P_n = 1 - (1 - P_{n-1}^2)^2$. (Lower panel) Example of an automaton obtained during learning from a prefix tree acceptor (PTA). It exhibits a series–parallel pattern, if not a self-similar one.

on, possibly towards an accepting state, or it does not match and the string is not accepted. When the edges appear on the same branch, the string must satisfy all conditions. This acts as an AND logical door, or as a Min node in a two-player game (see Figure B.1, upper panel). If we assume that the probability that the string successfully passes an elementary test is P_{n-1} then the probability that the string successfully passes the series of b tests is $Q_n = P_{n-1}^b$.

However, if several branches, say b, are available in parallel (see Figure B.2) and if the probability that the string passes one branch is Q_n then the probability that the string does not pass the parallel circuit is $1 - P_n = (1 - Q_n)^b$.

Thus, we obtain again $P_n = 1 - (1 - P_{n-1}^b)^b$. Overall, the probability that a string passes a self-similar circuit such as that of Figure B.2 has the same form as

Figure B.1. As the depth of the circuit increases, the transition from probability 0 to probability 1 sharpens and tends towards a step function.

Therefore, if an automaton exhibited a topology akin to such self-similar series-parallel circuits, should one expect a phase-transition-like profile for the coverage of random strings? Is this realistic, or is it only a game of the mathematical mind? This remains to be studied more carefully.

References

ACHLIOPTAS, D., KIROUSIS, L., KRANAKIS, E., KRINZAC, D., MOLLOY, M., and STAMATIOU, Y. (1997) Random constraint satisfaction: a more accurate picture. In *Lecture Notes in Computer Science*, vol. **1330**, pp. 107–120.

ACHLIOPTAS, D., KIROUSIS, L., KRANAKIS, E., KRIZANC, D., MOLLOY, M., and STAMATIOU, Y. (2001a) Random constraint satisfaction: a more accurate picture. *Constraints* **6**: 329–344.

ACHLIOPTAS, D., KIROUSIS, L., KRANAKIS, E., and KRIZANC, D. (2001b) Rigorous results for random $(2 + p)$-SAT. *Theor. Computer Sci.* **265**: 109–129.

ALAVA, M., ARDELIUS, J., AURELL, E., KASKI, P., KRISHNAMURTHY, S., and ORPONEN, P. (2008) Circumspect descent prevails in solving random constraint satisfaction problems. *PNAS* **105**: 15 253–15 257.

ALEKSIEJUK, A., HOLYST, A., and STAUFFER, D. (2002) Ferromagnetic phase transition in Barabási–Albert networks. *Physica A* **310**: 260–266.

ALPHONSE, E. (2009) Empirical study of the phase transition in relational learning. In *Proc. CAP 09*, pp. 173–184, Hammamet.

ALPHONSE, E., and OSMANI, A. (2007) Phase transition and heuristic search in relational learning. In *Proc. 6th Int. Conf. on Machine Learning and Applications*, pp. 112–117, Cincinnati, Ohio.

ALPHONSE, E., and OSMANI, A. (2008a) A model to study phase transition and plateaus in relational learning. In *Lecture Notes in Computer Science*, vol. **5194**, pp. 6–23.

ALPHONSE, E., and OSMANI, A. (2008b) On the connection between the phase transition of the covering test and the learning success rate in ILP. *Machine Learning* **70**: 135–150.

ALPHONSE, E., and OSMANI, A. (2009) Empirical study of relational learning algorithms in the phase transition framework. In *Lecture Notes in Computer Science*, vol. **5781**, pp. 51–66.

ALPHONSE, E., and ROUVEIROL, C. (2000) Lazy propositionalisation for relational learning. In *Proc. 14th European Conf. on Artificial Intelligence*, pp. 256–260, Berlin.

ANDRECUT, M., and KAUFFMAN, S. (2010) Phase transition in non-linear random networks. *arXiv:1003.0871v2*.

ANGLANO, C., and BOTTA, M. (2002) Now G-net: Learning classification programs on networks of workstations. *IEEE Trans. Evolutionary Comput.* **6**: 463–480.

ANGLANO, C., GIORDANA, A., LOBELLO, G., and SAITTA, L. (1997) A network genetic algorithm for concept learning. In *Proc. 7th Int. Conf. on Genetic Algorithms*, pp. 434–441, East Lansing, MI.

ANGLANO, C., GIORDANA, A., LOBELLO, G., and SAITTA, L. (1998) An experimental evaluation of coevolutive concept learning. In *Proc. 15th Int. Conf. on Machine Learning*, pp. 19–23, Madison, WI.

APT, K. (2003) *Principles of Constraint Programming*. Cambridge University Press.

APT, K., and WALLACE, M. (2007) *Constraint Logic Programming using Eclipse*. Cambridge University Press.

ARENAS, A., DÍAZ, A., KRTHS, J., MORENO, Y., and ZHOU, C. (2008) Synchronization in complex networks. *Phys. Rep.* **469**: 93–153.

AXELROD, R. (1997) The dissemination of culture – a model with local convergence and global polarization. *J. Conflict Resolution* **41**: 203–226.

BACCHUS, F., and VAN BEEK, P. (1998) On the conversion between non-binary and binary constraint satisfaction problems. In *Proc. 15th Nat. Conf. on Artificial Intelligence*, pp. 311–318, Madison, WI.

BARABÁSI, A., and ALBERT, R. (1999) Emergence of scaling in random networks. *Science* **159**: 509–512.

BARABÁSI, A., and BONABEAU, E. (2003) Scale-free networks. *Scientific American* **288**: 50–59.

BARRÉ, J., CIANI, A., FANELLI, D., BAGNOLI, F., and RUFFO, S. (2009) Finite size effects for the Ising model on random graphs with varying dilution. *Physica A* **388**: 3413–3425.

BARRAT, A., and WEIGT, M. (2000) On the properties of small-world networks. *Eur. Phys. J. B* **13**: 547–560.

BASAR, E. (1983) Toward a physical approach to integrative physiology: brain dynamics and physical causality. *Amer. J. Physiol.* **245**: 510–533.

BASKIOTIS, N., and SEBAG, M. (2004) C4.5 competence map: a phase transition-inspired approach. In *Proc. Int. Conf. on Machine Learning*, pp. 73–80, Banff.

BERGADANO, F., and GIORDANA, A. (1988) A knowledge intensive approach to concept induction. In *Proc. 5th Int. Conf. on Machine Learning*, pp. 305–317, Ann Arbor, MI.

BERGADANO, F., GIORDANA, A., and SAITTA, L. (1988) Learning concepts in noisy environment. *IEEE Tran. Pattern Analysis and Machine Intelligence* **PAMI-10**: 555–578.

BESSIÈRE, C., HEBRARD, E., and O'SULLIVAN, B. (2009) Minimising decision tree as combinatorial optimisation. In *Proc. 15th Int. Conf. on Principles and Practice of Constraint Programming*, pp. 173–187, Lisbon.

BETHE, H. (1935) Statistical theory of superlattices. *Proc. Roy. Soc. London Ser. A* **150**: 552–575.

BIANCONI, G. (2009) The entropy of network ensembles. *Phys. Rev. E* **79**: 036 114.

BIEHL, M., AHR, M., and SCHLÄSSER, E. (2000) Statistical physics of learning: phase transitions in multilayered neural networks. *Adv. Solid State Phys.* **40**: 819–826.

BIELY, C., and THURNER, S. (2006) Statistical mechanics of scale-free networks at a critical point: complexity without irreversibility? *Phys. Rev. E* **74**: 066 116.

BIERE, A., HEULE, M., VAN MAAREN, H., and WALSH, T. (2009) *Handbook of Satisfiability: Frontiers in Artificial Intelligence, Vol. 185*. IOS Press, Amsterdam.

BINDER, K., and LUIJTEN, E. (2001) Monte Carlo tests of renormalization-group predictions for critical phenomena in Ising models. *Phys. Rep.* **344**: 179–253.

BIROLI, G., MONASSON, R., and WEIGT, M. (2000) A variational description of the ground state structure in random satisfiability problems. *Eur. Phys. J. B.* **14**: 551–574.

BLYTHE, R., and EVANS, M. (2003) The Lee–Yang theory of equilibrium and nonequilibrium phase transitions. *Brazilian J. Phys.* **33**: 464–475.

BOTTA, M., and GIORDANA, A. (1993) SMART+: a multi-strategy learning tool. In *Proc. 13th Int. Joint Conf. on Artificial Intelligence*, pp. 937–943, Chambéry.

BOTTA, M., GIORDANA, A., and SAITTA, L. (1999) An experimental study of phase transitions in matching. In *Proc. 16th Int. Joint Conf. on Artificial Intelligence*, pp. 1198–1203, Stockholm.

BOTTA, M., GIORDANA, A., SAITTA, L., and SEBAG, M. (2000) Relational learning: hard problems and phase transitions. In *Lecture Notes in Computer Science*, vol. **1792**, pp. 178–189.

BOTTA, M., GIORDANA, A., SAITTA, L., and SEBAG, M. (2003) Relational learning as search in a critical region. *J. Machine Learning Res.* **4**: 431–463.

BOUFKHAD, Y., and DUBOIS, O. (1999) Length of prime implicants and number of solutions of random CNF formulas. *Theor. Computer Sci.* **215**: 1–30.

BOURNAUD, I., COURTINE, M., and ZUCKER, J.-D. (2003) Propositionalization for clustering symbolic relational descriptions. In *Proc. 12th Int. Conf. on Inductive Logic Programming*, pp. 1–16, Szeged.

BRACHMAN, R., and LEVESQUE, H. (2004) *Knowledge Representation and Reasoning*. Morgan Kaufmann.

BRASSARD, G., and BRATLEY, P. (1988) *Algorithmics: Theory and Practice*. Prentice Hall.

BRAUNSTEIN, A., PAGNANI, A., WEIGT, M., and ZECCHINA, R. (2008) Inference algorithms for gene networks: a statistical mechanics analysis. *J. Stat. Mech.*, 12 001.

BREIMAN, L., FRIEDMAN, J., OHLSEN, R., and STONE, C. (1984) *Classification And Regression Trees*. Wadsworth & Brooks.

BRUNSON, T., and BOETTCHER, S. (2009) The peculiar phase transitions of the Ising model on a small-world network. In *Proc. 76th Annual Meeting of the Southeastern Section of APS*, p. BAPS.2009.SES.NA.9, Atlanta, Georgia.

CARAMIA, M., and DELL'OLMO, P. (2002) Constraint propagation in graph coloring. *J. Heuristics* **8**: 83–107.

CARMESIN, H., and ARNDT, S. (1995) Organization of motion percepts. University of Bremen, ZKW Bericht no. 6/95.

CASEAU, Y. (1991) Abstraction interpretation of constraints on order-sorted domains. In *Proc. Int. Symp. on Logic Programming*, pp. 435–452, San Diego, CA.

CASTELLANO, C., FORTUNATO, S., and LORETO, V. (2009) Statistical physics of social dynamics. *Rev. Mod. Phys.* **81**: 591–646.

CASTELLÓ, X., EGUÍLUZ, V., and MIGUEL, M. S. (2006) Ordering dynamics with two non-excluding options: bilingualism in language competition. *New J. Physics* **8**: 308–322.

CHEESEMAN, P., KANEFSKY, B., and TAYLOR, W. (1991) Where the really hard problems are. In *Proc. 12th Int. Joint Conf. on Artificial Intelligence*, pp. 163–169, Sydney.

CHOMSKY, N. (1957) *Syntactic Structures*. Mouton.

CHURCH, A. (1936) An unsolvable problem of elementary number theory. *Ameri. J. Math.* **58**: 345–363.

CIPRA, B. (1987) An introduction to the Ising model. *Ameri. Math. Monthly* **94**: 937–959.

CIPRA, B. (2000) The Ising model is **NP**-complete. *SIAM News **33*** (6).

CLIFFORD, P., and SUDBURY, A. (1972) A model for spatial conflict. *Biometrika* **60**: 581–588.

COCCO, S., LEIBLER, S., and MONASSON, R. (2009) Neuronal couplings between retinal ganglion cells inferred by efficient inverse statistical physics methods. *PNAS* **106**: 14 058–14 062.

COHEN, W. W. (1995) Fast effective rule induction. In *Proc. Int. Conf. on Machine Learning*, pp. 115–123, Tahoe City, CA.

CONTUCCI, P., GALLO, I., and MENCONI, G. (2008) Phase transitions in social sciences: two-population mean field theory. *Int. J. Mod. Phys. B* **22**: 2199–2212.

COOK, S. (1971) The complexity of theorem-proving procedures. In *Proc. 3rd Annual ACM Symp. on Theory of Computing*, pp. 151–158, New York.

CORNUÉJOLS, A., and SEBAG, M. (2008) A note on phase transitions and computational pitfalls of learning from sequences. *Int. J. Intelligent Information Syst.* **31**: 177–189.

CORTES, C., and VAPNIK, V. (1995) Support vector machines. *Machine Learning* **20**: 273–297.

CRAWFORD, J., and AUTON, L. (1996) Experimental results on the crossover point in random 3-SAT. *Artificial Intelligence* **81**: 31–58.

DAVIS, M., and PUTNAM, H. (1960) A computing procedure for quantification theory. *J. ACM* **7**: 201–215.

DAVIS, M., LOGEMANN, G., and LOVELAND, D. (1962) A machine program for theorem-proving. *Commun. ACM* **5**: 394–397.

DE JONG, K. A., SPEARS, W. M., and GORDON, F. D. (1993) Using genetic algorithms for concept learning. *Machine Learning* **13**: 161–188.

DE LA HIGUERA, C. (2010) *Grammatical Inference: Learning Automata and Grammars*. Cambridge University Press.

DECHTER, R. (2003) *Constraint Processing*. Morgan Kaufmann.

DEINEKO, V. G., JONSSON, P., KLASSON, M., and KROKHIN, A. A. (2008) The approximability of MaxCSP with fixed-value constraints. *J. ACM* **55**.

DEQUEN, G., and DUBOIS, O. (2006) An efficient approach to solving random k-SAT problems. *J. Automated Reasoning* **37**: 261–276.

DIETRICH, R., OPPER, M., and SOMPOLINSKY, H. (1999) Statistical mechanics of support vector networks. *Phys. Rev. Lett.* **82**: 2975–2978.

DIETTERICH, T. (1998) Approximate statistical tests for comparing supervised classification learning algorithms. *Neural Computation* **10**: 1895–1923.

DIETTERICH, T., and MICHALSKI, R. (1983) A comparative review of selected methods for learning from examples. In *Machine Learning, an Artificial Intelligence Approach*, eds. J. Carbonell, R. Michalski, and T. Mitchell, pp. 41–81, Tioga.

DIETTERICH, T., LATHROP, R., and LOZANO-PEREZ, T. (1997) Solving the multiple-instance problem with axis-parallel rectangles. *Artificial Intelligence* **89**: 31–71.

DOROGOVTSEV, S., GOLTSEV, A., and MENDES, J. (2002) Ising model on networks with an arbitrary distribution of connections. *Phys. Rev. E* **66**: 016 104.

DUBOIS, O., and DEQUEN, G. (2001) A backbone-search heuristic for efficient solving of hard 3-SAT formulae. In *Proc. 17th Int. Joint Conf. on Artificial Intelligence*, pp. 248–253, Seattle, WA.

DURBIN, R., EDDY, S., KROGH, A., and MITCHISON, G. (1998) *Biological Sequence Analysis*. Cambridge University Press.

DURRETT, R. (2007) *Random Graph Dynamics*. Cambridge University Press.

EÉN, N., and SÖRENSSON, N. (2004) An extensible SAT solver. In *Lecture Notes in Computer Science*, vol. **2919**, pp. 333–336.

ELLMAN, T. (1993) Abstraction via approximate symmetry. In *Proc. Int. Joint Conf. on Artificial Intelligence*, pp. 916–921.

ENGEL, A., and VAN DEN BROECK, C. (2001) *Statistical Mechanics of Learning*. Cambridge University Press.

ERDÖS, P., and RÉNYI, P. (1959) On random graphs. *Publ. Math. Debrecen* **6**: 290–297.

ERENRICH, J., and SELMAN, B. (2003) Sampling combinatorial spaces using biased random walks. In *Proc. Int. Joint Conf. on Artificial Intelligence*, pp. 1376–1380, Acapulco.

FABRE-THORPE, M., DELORME, A., MARLOT, C., and THORPE, S. (2001) A limit to the speed of processing in ultra-rapid visual categorization of novel natural scenes. *J. Cognitive Neurosci.* **13**: 171–180.

FRANCO, J., and PAULL, M. (1983) Probabilistic analysis of the Davis–Putnam procedure for solving the satisfiability problem. *Discrete Applied Math.* **5**: 77–87.

FRANK, J., CHEESEMAN, P., and STUTZ, J. (1997) When gravity fails: local search topology. *J. Artificial Intelligence Res.* **17**: 249–281.

FREEMAN, W. (1975) *Mass Action in the Nervous System*. Academic Press.

FREEMAN, W. (1988) Why neural networks don't fly: inquiry into the neurodynamics of biological intelligence. In *Proc. 2nd Annual Int. Conf. on Neural Networks*, pp. 1–8, San Diego, CA.

FREEMAN, W. (1990) On the problem of anomalous dispersion in chao-to-chaotic phase transitions of neural masses, and its significance for the management of perceptual information in brains. In *Synergetics or Cognition*, eds. H. Haken and M. Stadler, pp. 126–143, Springer-Verlag.

FREEMAN, W. (2009) Vortices in brain activity: their mechanism and significance for perception. *Neural Networks* **22**: 491–501.

FREEMAN, W., and VITIELLO, G. (2006) Nonlinear brain dynamics as macroscopic manifestation of underlying many-body field dynamics. *Phys. Life Rev.* **3**: 93–118.

FREY, B. J., KSCHISCHANG, F. R., LOELIGER, H., and WIBERG, N. (1998) Factor graphs and algorithms. In *Proc. 35th Allerton Conf. on Communications, Control, and Computing*, pp. 666–680, Allerton, Iowa.

FRIEDMAN, E., and NISHIMURA, J. (2010) Explosive percolation in social and physical networks. *arXiv:1001.4772*.

FROST, D., and DECHTER, R. (1995) Look-ahead value ordering for constraint satisfaction problems. In *Proc. 14th Int. Joint Conf. on Artificial Intelligence*, pp. 572–578, Montreal.

FROST, D., RISH, I., and VILA, L. (1997) Summarizing CSP hardness with continuous probability distributions. In *Proc. 14th National Conference on Artificial Intelligence*, pp. 327–333, Providence, RI.

FU, Y., and ANDERSON, P. (1986) Application of statistical mechanics to **NP**-complete problems in combinatorial optimization. *J. Phys. A* **19**: 1605–1620.

FUCHS, A., DEECKE, L., and KELSO, J. (2000) Phase transition in the human brain revealed by large squid arrays: response to Daffertshofer, Peper and Beek. *Phys. Lett. A* **266**: 303–308.

GALAM, S. (2002) Minority opinion spreading in random geometry. *Eur. Phys. J. B* **25**: 403–406.

GALAM, S., GEFEN, Y., and SHAPIR, Y. (1982) Sociophysics: a mean behavior model for the process of strike. *J. Math. Sociol.* **9**: 1–13.

GAREY, M. R., and JOHNSON, D. S. (1979) *Computers and Intractability*: *A Guide to the Theory of* **NP**-*completeness*. W. H. Freeman and Co.

GAUDEL, R., SEBAG, M., and CORNUÉJOLS, A. (2008) A phase transition-based perspective on multiple instance kernels. *Lecture Notes in Computer Science*, vol. **4894**, pp. 112–121.

GENT, I., and WALSH, T. (1996) The TSP phase transition. *Artificial Intelligence* **88**: 349–358.

GENT, I. P., MACINTYRE, E., PROSSER, P., SMITH, B., and WALSH, T. (1995) Scaling effects in the CSP phase transition. In *Lecture Notes in Computer Science* vol. **976**, pp. 70–87.

GENT, I. P., MACINTYRE, E., PROSSER, P., SMITH, B., and WALSH, T. (2001) Random constraint satisfaction: flaws and structure. *Constraints* **6**: 345–372.

GIORDANA, A., and SAITTA, L. (1994) Learning disjunctive concepts by means of genetic algorithms. In *Proc. 11th Int. Conf. on Machine Learning*, pp. 96–104, New Brunswick, NJ.

GIORDANA, A., and SAITTA, L. (2000) Phase transitions in relational learning. *Machine Learning* **41** (2): 17–251.

GIORDANA, A., SAITTA, L., and BERGADANO, F. (1993) Enigma: a system that learns diagnostic knowledge. *IEEE Trans. Knowledge and Data Eng.* **KDE-5**: 15–28.

GIORDANA, A., NERI, F., SAITTA, L., and BOTTA, M. (1997) Integrating multiple learning strategies in first order logics. *Machine Learning* **27**: 209–240.

GIORDANA, A., SAITTA, L., SEBAG, M., and BOTTA, M. (2000a) Analyzing relational learning in the phase transition framework. In *Proc. 17th Int. Conf. on Machine Learning*, pp. 311–318, Stanford, CA.

GIORDANA, A., SAITTA, L., SEBAG, M., and BOTTA, M. (2000b) Can relational learning scale up? In *Lecture Notes in Computer Science*, vol. **1932**, pp. 85–104.

GITTERMAN, M., and HALPERN, V. (2004) *Phase Transitions: A Brief Account with Modern Applications*. World Scientific.

GÖDEL, K. (1931) Über formal unentscheidbare Sätze der Principia Mathematica und verwandter Systeme. *Monatshefte Math. Phys.* **38**: 173–198.

GOLD, E. M. (1967) Language identification in the limit. *Information and Control* **10**: 447–474.

GOLDBERG, A. (1979) On the complexity of satisfiability problem. Courant Computer Science Report no. 16, New York University.

GOLDBERG, D. (1989) *Genetic Algorithms*. Addison-Wesley.

GOMES, C., and SELMAN, B. (1997) Problem structure in the presence of perturbations. In *Proc. 14th Nat. Conf. on Artificial Intelligence*, pp. 431–437, Providence, RI.

GOTTLOB, G., LEONE, N., and SCARSELLO, F. (1997) On the complexity of some inductive logic programming problems. In *Lecture Notes in Computer Science*, vol. **1297**, pp. 17–32.

GUPTA, D. K. (2000) Dynamic variable ordering in graph based backjumping algorithms for *CSPs*. *Int. J. Computer Math.* **75**: 167–186.

GUSSFIELD, D. (1997) *Algorithms on Strings, Trees, and Sequences*. Cambridge University Press.

HAKEN, H. (2002) *Brain Dynamics*. Springer.

HARALICK, R. M., and ELLIOTT, G. L. (1980) Increasing tree search efficiency for constraint satisfaction problems. *Artificial Intelligence* **14**: 263–313.

HARTMANN, A., and WEIGT, M. (2005) *Phase Transitions in Combinatorial Optimization Problems*. Wiley-VCH.

HAUSSLER, D., KEARNS, M., SEUNG, H., and TISHBY, N. (1994) Rigorous learning curve bounds from statistical mechanics. *Machine Learning* **25**: 195–236.

HERRERO, C. (2002) Ising model in small-world networks. *Phys. Rev. E* **65**: 066110.

HERRERO, C. (2009) Antiferromagnetic Ising model in scale-free networks. *Eur. Phys. J. B* **70**: 435–441.

HEULE, M., DUFOUR, M., VAN MAAREN, H., and VAN ZWIETEN, J. (2004) March_eq: implementing efficiency and additional reasoning into a lookahead SAT solver. *J. Satisfiability, Boolean Modeling and Computation* **1**: 25–30.

HIRSCH, E. A., and KOJEVNIKOV, A. (2005) Unitwalk: a new SAT solver that uses local search guided by unit clause elimination. *Ann. Math. Artificial Intelligence* **43**: 91–111.

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

HOGG, T., HUBERMAN, B. A., and WILLIAMS, C., eds. (1996) *Frontiers in Problem Solving: Phase Transitions and Complexity. Artificial Intelligence* **81** (1–2).

HOLLAND, J. (1986) Escaping brittleness: the possibilities of general purpose learning algorithms applied to parallel rule based systems. In *Machine Learning: An Artificial Intelligence Approach*, eds. R. Michalski, J. Carbonell, and T. Mitchell, volume 2, Morgan Kauffman.

HOLTE, R., ACKER, L., and PORTER, B. (1989) Concept learning and the problem of small disjuncts. In *Proc. 11th Int. Joint Conf. on Artificial Intelligence*, pp. 813–818, Detroit, MI.

HOPCROFT, J., and ULLMAN, J. (1969) *Formal Languages and Their Relation to Automata*. Addison-Wesley.

HOWE, A., and LEVY, W. (2007) A hippocampal model predicts a fluctuating phase transition when learning certain trace conditioning paradigms. *Cognitive Neurodynamics* **1**: 143–155.

HUTTER, F., TOMPKINS, D., and HOOS, H. (2002) Scaling and probabilistic smoothing: efficient dynamic local search for SAT. In *Proc. 8th Int. Conf. on Principles and Practice of Constraint Programming*, pp. 233–248, London.

ISING, E. (1925) Beitrag zur Theorie des Ferromagnetismus. *Z. Physik* **31**: 253 258.

JAEGER, G. (1998) The Ehrenfest classification of phase transitions : introduction and evolution. *Arch. Hist. Exact Sci.* **53**: 51–81.

JAFFAR, J., and MAHER, M. (1994) Constraint logic programming: a survey. *J. Logic Programming* **19/20**: 503–581.

JEONG, D., HONG, H., KIM, B., and CHOL, M. (2003) Phase transition in the Ising model on a small-world network with distance-dependent interactions. *Phys. Rev. E* **68**: 027 101.

KAC, M., and WARD, J. (1952) A combinatorial solution of the two-dimensional Ising model. *Phys. Rev.* **88**: 1332–1337.

KAMATH, A., KARMARKAR, N., RAMAKRISHNAN, K., and RESENDE, M. (1992) A continuous approach to inductive inference. *Math. Programming* **57**: 215–238.

KARP, R., and PEARL, J. (1983) Searching for an optimal path in a tree with random costs. *Artificial Intelligence* **21**: 99–117.

KARUNARATNE, T., and BOSTRÖM, H. (2009) Graph propositionalization for random forests. In *Proc. Int. Conf. on Machine Learning and Applications*, pp. 196–201, Miami Beach, FL.

KAUTZ, H., and SELMAN, B. (1996) Pushing the envelope: planning, propositional logic, and stochastic search. In *Proc. 13th Nat. Conf. on Artificial Intelligence*, pp. 1194–1201, Portland, Oregon.

KEARNS, M., and VAZIRANI, U. (1994) *An Introduction to Computational Learning Theory*. MIT Press.

KEESOM, W. H., and VAN DEN ENDE, J. (1932) *Proc. Kon. Akad. Amsterdam* **35**: 743.

KELSO, J., DEL COLLE, J., and SCHÖNER, G. (1990) Action-perception as a pattern formation process. In *Proc. 13th Conf. on Attention and Performance*, pp. 139–169, Hillsdale, NJ.

KILBY, P., SLANEY, J., THIÉBAUX, S., and WALSH, T. (2005) Backbones and backdoors in satisfiability. In *Proc. 20th Nat. Conf. on Artificial Intelligence*, pp. 1368–1373, Pittsburgh, PA.

KING, R., SRINIVASAN, A., and STENBERG, M. (1995) Relating chemical activity to structure: an examination of ILP successes. *New Generation Computing* **13**: 411–433.

KINZEL, W. (1998) Phase transitions of neural networks. In *Proc. Minerva Workshop on Mesoscopics, Fractals and Neural Networks*, pp. 1455–1477, Eilat.

KIRKPATRICK, S., and SELMAN, B. (1994) Critical behavior in the satisfiability of random boolean expressions. *Science* **264**: 1297–1301.

KIRKPATRICK, S., GELATT, C. D., and VECCHI, M. P. (1983) Optimization by simulated annealing. *Science* **220**: 671–680.

KOK, S., and DOMINGOS, P. (2010) Learning Markov logic networks using structural motifs. In *Proc. 27th Int. Conf. on Machine Learning*, pp. 502–509, Haifa.

KOMIN, N., and TORAL, R. (2010) Phase transitions induced by microscopic disorder: a study based on the order parameter expansion. *arXiv:1003.1061*.

KOPPES, S. (2008) Cowan discovers analogy between behavior of chemical reaction networks, neural networks of brain. *Univ. Chicago Chron.* ***17*** (12).

KOWALSKI, R. (1979) *Logic for Problem Solving*. North-Holland.

KRAMER, S., LAVRAČ, N., and FLACH, P. (2000) Propositionalization approaches to relational data mining. In *Relational Data Mining*, eds. N. Lavrač and P. Flach, pp. 262–286, Springer-Verlag.

KRAMERS, H., and WANNIER, G. (1941) Statistics of the two-dimensional ferromagnet, I. *Phys. Rev.* **60**: 252–262.

KROC, L., SABHARWAL, A., GOMES, C., and SELMAN, B. (2009) Integrating systematic and local search paradigms: a new strategy for maxsat. In *Proc. IJCAI 2009*, pp. 544–551.

KROGEL, M., RAWLES, S., ŽELEZNÝ, F., FLACH, P., LAVRAC, N., and WROBEL, S. (2003) Comparative evaluation of approaches to propositionalization. In *Lecture Notes in Computer Sciences*, vol. **2835**, pp. 197–214.

KRUSE, P., CARMESIN, H., PAHLKE, L., STRÜBER, D., and STADLER, M. (1996) Continuous phase transitions in the perception of multistable visual patterns. University of Bremen, Germany: ZKW Bericht no. 3/95.

KRZCAKALA, F. (2005) How many colors to color a random graph? Cavity, complexity, stability and all that. *Prog. Theoret. Phys.* **157**: 357–360.

KSCHISCHANG, F., MEMBER, S., FREY, B. J., and ANDREA LOELIGER, H. (2001) Factor graphs and the sum–product algorithm. *IEEE Trans. Information Theory* **47**: 498–519.

KSHIVETS, O. (2008) Synergetics, artificial intelligence, and complex system analysis in recognition of phase transition early-invasive lung cancer. *J. Clin. Oncol.* **26**: 22183.

KUMAR, V. (1992) Algorithms for constraint satisfaction problems: a survey. *AI Magazine* **13**: 32–44.

LANDAU, L., and LIFSHITZ, E. (1976) *Mechanics, Third Edition*. Pergamon Press.

LANDAU, L., and LIFSHITZ, E. (1980) *Statistical Physics, Third Edition*. Pergamon Press.

LANG, K. J., PEARLMUTTER, B. A., and PRICE, R. A. (1998) Results of the Abbadingo One DFA learning competition and a new evidence-driven state merging algorithm. In *Lecture Notes in Computer Science*, vol. **1433**, pp. 1–12.

LANGLEY, P. (1986) Editorial: On machine learning. *Machine Learning* **1**: 5–10.

LAVRAČ, N., and DŽEROSKI, S. (1994) *Inductive Logic Programming: Techniques and Applications*. Ellis Horwood.

LAWLER, E. L., and WOOD, D. E. (1966) Branch-and-bound methods: a survey. *Op. Res.* **13**: 699–719.

LAWNICZAK, A., and TANG, X. (2005) Network traffic behaviour near phase transition point. *Eur. Phys. J. B* **50**: 231–236.

LEE, T., and YANG, C. (1952) Statistical theory of equations of state and phase transitions. II. Lattice gas and Ising model. *Phys. Rev. Lett.* **87**: 410–419.

LEHN, K. V., and BALL, W. (1987) A version space approach to learning context-free grammars. *Machine Learning J.* **2**: 39–74.

LEONE, M., VÁZQUEZ, A., R., and ZECCHINA, A. V. (2002) Ferromagnetic ordering in graphs with arbitrary degree distribution. *Eur. Phys. J. B* **28**: 191–197.

LEWENSTEIN, M., NOWAK, A., and LATANÉ, B. (1992) Statistical mechanics of social impact. *Phys. Rev. A* **45**: 763–776.

LI, C., MANYÀ, F., and PLANES, J. (2007a) New inference rules for Max-SAT. *J. Artificial Intelligence Res.* **30**: 321–359.

LI, C., WEY, W., and ZHANG, H. (2007b) Combining adaptive noise and look-ahead in local search for SAT. In *Lecture Notes in Computer Science*, vol. 4501, pp. 121–133.

MACINTYRE, E., PROSSER, P., SMITH, B., and WALSH, T. (1998) Random constraint satisfaction: theory meets practice. In *Lecture Notes in Computer Science*, vol. **1520**, pp. 325–339.

MALOBERTI, J., and SEBAG, M. (2004) Fast θ-subsumption with constraint satisfaction algorithms. *Machine Learning* **55**: 137–174.

MALZAHN, D., and OPPER, M. (2005) A statistical physics approach for the analysis of machine learning algorithms on real data. *J. Statist. Mech.: Theory and Experiment* **11**: 11 001–11 033.

MANDERIK, B., and SPIESSENS, P. (1989) Fine-grained parallel genetic algorithms. In *Proc. 4th Int. Conf. on Genetic Algorithms*, pp. 264–270, Fairfax, VA.

MARBUKH, V. (2007) Towards understanding of complex communication networks: performance, phase transitions and control. In *Proc. Workshop on Math. Performance Modeling and Analysis*, San Diego, CA.

MARQUES-SILVA, J., and MANQUINHO, V. (2008) Towards more effective unsatisfiability-based maximum satisfiability algorithms. In *Lecture Notes in Computer Science*, vol. 4996, pp. 225–230.

MARQUES-SILVA, J., and SAKALLAH, K. (1996) Grasp – a new search algorithm for satisfiability. In *Proc. IEEE/ACM Int. Conf. on Computer-Aided Design*, pp. 220–227, Washington, DC.

MARQUES-SILVA, J., and SAKALLAH, K. (1999) Grasp: a search algorithm for propositional satisfiability. *IEEE Trans. Computers* **48**: 506–521.

MARTIN, O., MONASSON, R., and ZECCHINA, R. (2001) Statistical mechanics methods and phase transitions in optimization problems. *Theoret. Computer Sci.* **265**: 3–67.

MATOLCSI, T. (1996) On the classification of phase transitions. *Zeit. Angewandte Math. Phys.* **47**: 837–857.

MAURO, N. D., BASILE, T., FERILLI, S., and ESPOSITO, F. (2010) Approximate relational reasoning by stochastic propositionalization. *Stud. Computational Intelligence* **265**: 81–109.

MERTENS, S., MÉZARD, M., and ZECCHINA, R. (2006) Threshold values of random k-SAT from the cavity method. *Random Structures and Algorithms* **28**: 340–373.

MÉZARD, M., and PARISI, G. (1985) Replicas and optimization. *J. de Phys. Lett.* **46**: 771–778.

MÉZARD, M., and PARISI, G. (1991) Replica field theory for random manifolds. *J. de Physique I* **1**: 809–836.

MÉZARD, M., and ZECCHINA, R. (2002) The random k-satisfiability problem: from an analytic solution to an efficient algorithm. *Phys. Rev. E* **66**: 56–126.

MÉZARD, M., PARISI, G., and VIRASORO, M. (1987) *Spin Glass Theory and Beyond*. World Scientific.

MÉZARD, M., MORA, T., and ZECCHINA, R. (2005) Clustering of solutions in the random satisfiability problem. *Phys. Revi. Lett.* **94**: 197 205.

MICHALEWICZ, Z. (1996) *Genetic Algorithms + Data Structures = Evolution Programs, Third Edition*. Springer.

MICHALSKI, R. (1983) A theory and methodology of inductive learning. In *Machine Learning: An Artificial Intelligence Approach*, eds. R. Michalski, J. Carbonell, and T. Mitchell, pp. 83–134. Morgan Kaufmann.

MICHARD, Q., and BOUCHAUD, J. (2005) Theory of collective opinion shifts: from smooth trends to abrupt swings. *Eur. Phys. J. B* **47**: 151–159.

MIHALKOVA, L., and MOONEY, R. (2007) Bottom-up learning of Markov logic network structure. In *Proc. 24th Int. Conf. on Machine Learning*, pp. 625–632, Corvallis, OR.

MINSKY, M., and PAPERT, S. (1969) *Perceptrons*. MIT Press.

MITCHELL, T. (1977) Version spaces: a candidate elimination approach to rule learning. In *Proc. 5th Int. Joint Conf. on Artificial Intelligence*, pp. 305–310, Cambridge, MA.

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

MITCHELL, T., and SCHWENZER, G. (1978) Applications of artificial intelligence for chemical inference: a computer program for automated empirical 13C-NMR rule formation. *Organic Magnetic Resonance* **11**: 378–384.

MONASSON, R. (2002) Threshold phenomena and complexity: a statistical physics analysis of the random satisfiability problem. In *Lectures on Random Combinatorial Problems*, MDSU, Lansing, USA.

MONASSON, R., and ZECCHINA, R. (1997) Statistical mechanics of the random k-SAT problem. *Phys. Rev. E* **56**: 1357–1361.

MONASSON, R., ZECCHINA, R., KIRKPATRICK, S., SELMAN, B., and TROYANSKY, L. (1999) 2+ p-sat: relation of typical-case complexity to the nature of the phase transition. *Random Structures and Algorithms* **15**: 414–435.

MONTANARI, A., RICCI-TERSENGHI, F., and SEMERJIAN, G. (2008) Clusters of solutions and replica symmetry breaking in random k-satisfiability. *J. Stat. Mech.: Theory and Experiment* p. P04004.

MOONEY, R. (1995) Encouraging experimental results on learning CNF. *Machine Learning* **19**: 79–92.

MOSKEWICZ, M., MADIGAN, C., ZHAO, Y., ZHANG, L., and MALIK, S. (2001) Chaff: engineering an efficient SAT solver. In *Proc. 38th Design Automation Conf.*, pp. 530–535, Las Vegas, NV.

MUGGLETON, S. (1991) Inductive logic programming. *New Generation Computing* **8**: 295–318.

MUGGLETON, S., ed. (1992) *Inductive Logic Programming*. Academic Press.

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

MUGGLETON, S., and BUNTINE, W. (1988) Machine invention of first-order predicates by inverting resolution. In *Proc. 5th Int. Conf. on Machine Learning*, pp. 339–352, Ann Arbor, Michigan.

MUGGLETON, S., and DE RAEDT, L. (1994) Inductive logic programming: theory and methods. *J. Logic Program.* **19**: 629–679.

MUGGLETON, S., and FENG, C. (1990) Efficient induction of logic programs. In *Proc. 1st Conf. on Algorithmic Learning Theory*, pp. 368–381, Tokyo.

MULET, R., PAGNANI, A., WEIGT, M., and ZECCHINA, R. (2002) Coloring random graphs. *Phys. Rev. Lett.* **89**: 268 701.

MURPHY, G. (2002) *The Big Book of Concepts*. MIT Press.

NACZAR, B., ZUCKER, J., HENEGAR, C., and SAITTA, L. (2007) Feature construction from synergic pairs to improve microarray-based classification. *Bioinformatics* **23**: 2866–2872.

NEWMAN, M. E. J. (2003) The structure and function of complex networks. *SIAM Review* **45**: 167 256.

NICOLAU, M., and SCHOENAUER, M. (2009) On the evolution of free-scale topologies with a gene regulatory network model. *Biosystems* **98**: 137–148.

NISHIMORI, H. (2001) *Statistical Physics of Spin Glasses and Information Processing*. Oxford University Press.

OHIRA, T., and SAWATARI, R. (1998) Phase transition in computer network traffic model. *Phys. Rev. E* **58**: 193–195.

ONCINA, J., and GARCÍA, P. (1992) Inferring regular languages in polynomial updated time. In *Pattern Recognition and Image Analysis : Selected Papers from the 4th Spanish Symposium*, pp. 49–61, World Scientific.

ONSAGER, L. (1944) Crystal statistics I. A two-dimensional model with an order–disorder transition. *Phys. Rev.* **65**: 117–149.

PALMER, E. (1985) *Graphical Evolution*. Wiley.

PARK, J., and NEWMAN, M. (2004) The statistical mechanics of networks. *Phys. Rev. E* **70**: 066 117.

PEARL, J. (1984) *Heuristics. Intelligent Search Strategies for Computer Problem Solving*. Addison-Wesley.

PEARL, J. (1988) *Probabilistic Reasoning in Intelligent Systems, second edition*. Morgan Kauffmann.

PEIERLS, R. (1936) Ising's model of ferromagnetism. *Proc. Camb. Phil. Soc.* **32**: 477–481.

PELALSKI, A. (2000) Ising model on a small world network. *Phys. Rev. E* **64**: 057 104.

PERCUS, A., OSTRATE, G., and MOORE, C., eds. (2006) *Computational Complexity and Statistical Physics*. Oxford University Press.

PERNOT, N., CORNUÉJOLS, A., and SEBAG, M. (2005) Phase transitions within grammatical inference. In *Proc. 19th Int. Joint Conf. on Artificial Intelligence*, pp. 811–816, Edinburgh.

PIPPARD, A. (1957) *Elements of Classical Thermodynamics*. Cambridge University Press.

PLOTKIN, G. (1970) A note in inductive generalization. In *Machine Intelligence*, eds. B. Meltzer and D. Michie, vol. 5, pp. 153–163.

PROSSER, P. (1993) Hybrid algorithms for the constraint satisfaction problem. *Computational Intelligence* **9**: 268 – 269.

PROSSER, P. (1995) MAC-CBJ: Maintaining arc consistency with conflict-directed back-jumping. Research Report 95/177, University of Strathclyde.

PROSSER, P. (1996) An empirical study of phase transitions in constraint satisfaction problems. *Artificial Intelligence* **81**: 81–109.

PURDOM, P. J., and BROWN, C. (1987) Polynomial average-time satisfiability problems. *Information Science* **41**: 23–42.

QUINLAN, J. (1993) *C4.5: Programs for Machine Learning*. Morgan Kaufmann.

QUINLAN, J., and CAMERON-JONES, R. (1993) FOIL: a midterm report. In *Lecture Notes in Artificial Intelligence*, vol. **667**, pp. 3–20.

QUINLAN, R. (1986) Induction of decision trees. *Machine Learning* **1**: 81–106.

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

RABINER, L. (1989) A tutorial on hidden Markov models and selected applications in speech recognition. *Proc. IEEE* **77**: 257–286.

RAYWARD-SMITH, V., OSMAN, I., REEVES, C., and SMITH, G. (1989) *Modern Heuristic Search Methods*. Wiley.

RISSANEN, J. (1978) Modeling by shortest data description. *Automatica* **14**: 465–471.

ROSCH, E. (1973) Natural categories. *Cognitive Psychology* **4**: 328–350.

ROSENBLATT, F. (1958) The perceptron: a probabilistic model for information storage and organization in the brain. *Psychol. Rev.* **65**: 386–407.

ROZENFELD, H., SONG, C., and MAKSE, H. (2010) Small-world to fractal transition in complex networks: a renormalization group approach. *Phys. Rev. Lett.* **104**: 025 701.

RÜCKERT, U., and KRAMER, S. (2003) Stochastic local search in k-term DNF-learning. In *Proc. Int. Conf. on Machine Learning*, pp. 648–655, Washington, DC.

RÜCKERT, U., KRAMER, S., and DERAEDT, L. (2002) Phase transitions and stochastic local search in k-term DNF learning. In *Lecture Notes in Computer Science*, vol. **2430**, pp. 43–63.

RUMELHART, D. E., and MCCLELLAND, J. L. (1986) *Parallel Distributed Processing: Explorations in the Microstructure of Cognition, Parts I & II*. MIT Press.

RUMELHART, D. E., HINTON, G. E., and WILLIAMS, R. J. (1986) Learning representations by back-propagating errors. *Nature* **323**: 533–536.

RUSSELL, S., and NORVIG, P. (2003) *Artificial Intelligence: A Modern Approach*. Prentice Hall.

SAITTA, L., and ZUCKER, J. (2000) Abstraction and phase transitions in relational learning. In *Lecture Notes in Artificial Intelligence*, vol. **1864**, pp. 291–301.

SAITTA, L., and ZUCKER, J. (2001) A model of abstraction in visual perception. *Int. J. Applied Intelligence* **80**: 134–155.

SAITTA, L., and ZUCKER, J. (2009) Perception-based granularity levels in concept representation. In *Lecture Notes in Computer Science*, vol. **1932**, pp. 201–214.

SCHEFFER, T., HERBRICH, R., and WYSOTZKI., F. (1996) Efficient θ-subsumption based on graph algorithms. In *Lecture Notes in Artificial Intelligence*, vol. **1314**, pp. 212–228.

SCHRAG, R., and MIRANKER, D. (1996) Abstraction and the CSP phase transition boundary. In *Proc. Int. Conf. on Artificial Intelligence and Mathematics*, pp. 138–141, Fort Lauderdale, FL.

SCHUURMANS, D., and SOUTHEY, F. (2001) Local search characteristics of incomplete sat procedures. *Artificial Intelligence* **132**(2): 121–150.

SEBAG, M., and ROUVEIROL, C. (1997) Tractable induction and classification in first order logic via stochastic matching. In *Proc. 15th Int. Joint Conf. on Artificial Intelligence*, pp. 888–893, Nagoya.

SEBAG, M., and ROUVEIROL, C. (2000) Resource-bounded relational reasoning: induction and deduction through stochastic matching. *Machine Learning* **38**: 41–62.

SELMAN, B., LEVESQUE, H., and MITCHELL, D. (1992) A new method for solving hard satisfiability problems. In *Proc. 11th Nat. Conf. on Artificial Intelligence*, pp. 440–446, Washington, DC.

SEMERJIAN, G., and MONASSON, R. (2003) Relaxation and metastability in a local search procedure for the random satisfiability problem. *Phys. Rev. E* **67**: 066 103.

SERRA, A., GIORDANA, A., and SAITTA, L. (2001) Learning on the phase transition edge. In *Proc. 17th Int. Joint Conf. on Artificial Intelligence*, pp. 921–926, Seattle, WA.

SEUNG, H., SOMPOLINSKY, H., and TISHBY, N. (1992) Statistical mechanics of learning from examples. *Phys. Rev. A* **45**: 6056–6091.

SHAWE-TAYLOR, J., and CRISTIANINI, N. (2004) *Kernel Methods for Pattern Analysis*. Cambridge University Press.

SHERRINGTON, D., and KIRKPATRICK, S. (1975) Solvable models of spin glass. *Phys. Rev. Lett.* **35**: 1792–1796.

SIMON, H. (1955) On a class of skew distribution functions. *Biometrika* **42**: 425–440.

SMITH, B. (1994) Locating the phase transition in binary constraint satisfaction problems. *Artificial Intelligence* **81**: 155–181.

SMITH, B. (2001) Constructing an asymptotic phase transition in random binary constraint satisfaction problems. *Theor. Computer Sci.* **265**: 265–283.

SMITH, B., and DYER, M. (1996) Locating the phase transition in binary constraint satisfaction problems. *Artificial Intelligence* **81**: 155–181.

SMITH, B., and GRANT, S. (1997) Modelling exceptionally hard constraint satisfaction problems. In *Lecture Notes in Computer Science*, vol. **1330**, pp. 182–195.

SOLOMONOFF, R., and RAPOPORT, A. (1951) Connectivity of random nets. *Bull. Math. Biophys.* **13**: 107–116.

SÖRENSSON, N., and EÉN, N. (2005) MiniSAT v1.13: a SAT solver with conflict clause minimization. In *Proc. 8th Int. Conf. on Theory and Applications of Satisfiability Testing,* St. Andrews.

SRINIVASAN, A., and MUGGLETON, S. (1995) Comparing the use of background knowledge by two ILP systems. In *Proc. Conf. on Inductive Logic Programming 1995*, Katholieke Universiteit Leuven.

SRINIVASAN, A., MUGGLETON, S., and KING, R. (1995) Comparing the use of background knowledge by two ILP systems. In *Proc. 5th Int. Workshop on ILP*, pp. 199–229, Leuven.

STRAUSS, D. (1986) On a general class of models for interaction. *SIAM Review* **28**: 513–527.

SUTTON, A., HOWE, A., and DARRELL, L. (2009) Estimating bounds on expected plateau size in MAX-SAT problems. In *Proc. 2nd Int. Workshop on*

Engineering Stochastic Local Search Algorithms. Designing, Implementing and Analyzing Effective Heuristics, pp. 31–45.

SVRAKIC, N. (1980) Critical temperatures of Ising models. *Phys. Lett. A* **80**: 43–44.

SZABO, B., SZOLLOSI, G., GONCI, B., JURANYI, Z., SELMECZI, D., and VICSEK, T. (2006) Phase transition in the collective migration of tissue cells: experiment and model. *Phys. rev. E* **74**: 061 908.

TÖNJES, R., MASUDA, N., and KORI, H. (2010) Synchronization transition of identical phase oscillators in a directed small-world network. *arXiv/1003.2020*.

TSANG, E. (1993) *Foundations of Constraint Satisfaction*. Academic Press.

TURING, A. (1936) On computable numbers, with an application to the entscheidungsproblem. *Proc. London Math. Soc. 2* **42**: 230–265.

TURNER, J. (1988) Almost all k-colorable graphs are easy to color. *J. Algorithms* **9**: 63–82.

VALIANT, L. (1984) A theory of the learnable. *Commun. ACM* **27**: 1134–1142.

VAN HENTENRYCK, P. (1989) *Constraint Satisfaction in Logic Programming*. MIT Press.

VAPNIK, V. N. (1995) *The Nature of Statistical Learning Theory*. Springer-Verlag.

VAPNIK, V. N. (1999) An overview of statistical learning theory. *IEEE Trans. Neural Networks* **10**: 988–999.

VÁZQUEZ, A., and WEIGT, M. (2003) Computational complexity arising from degree correlations in networks. *Phys. Rev. E* **67**: 027101.

VICSEK, T. (2007) Phase transitions and overlapping modules in complex networks. *Physica A* **378**: 20–32.

VILONE, D., and CASTELLANO, C. (2004) Solution of voter model dynamics on annealed small-world networks. *Phys. Rev. E* **69**: 016 109.

WALSH, T. (1999) Search in a small world. In *Proc. 16th Int. Joint Conf. on Artificial Intelligence*, pp. 1172–1177, Stockholm.

WÄSTLUND, J. (2009) Replica symmetry and combinatorial optimization. In *Proc. Conf. on Physics of Algorithms*, Santa Fe, New Mexico.

WATKIN, T., RAU, A., and BIEHL, M. (1993) The statistical mechanics of learning a rule. *Rev. Mod. Phys.* **65**: 499–556.

WATTS, D., and STROGATZ, S. (1998) Collective dynamics of *small-world* networks. *Nature* **393**: 440–442.

WERNER, G. (2007) Metastability, criticality and phase transitions in brain and its models. *Biosystems* **90**: 496–508.

WIECZOREK, S., BISSON, G., and GORDON, M. (2006) Guiding the search in the NO region of the phase transition problem with a partial subsumption test. In *Lecture Notes in Computer Science*, vol. **4212**, pp. 817–824.

WILLIAMS, C., and HOGG, T. (1994) Exploiting the deep structure of constraint problems. *Artificial Intelligence* **70**: 73–117.

WILLIAMS, R., GOMES, C., and SELMAN, B. (2003) Backdoors to typical case complexity. In *Proc. 18th Int. Joint Conf. on Artificial Intelligence*, pp. 1173–1178, Acapulco.

WINSTON, P. (1975) Learning structural descriptions from example. In *The Psychology of Computer Vision*, ed. P. Winston, pp. 157–209, McGraw-Hill.

WITOELAR, A., BIEHL, M., GHOSH, A., and HAMMER, B. (2008) Learning dynamics and robustness of vector quantization and neural gas. *Neurocomputing* **71**: 1210–1219.

WITTEN, I., and FRANK, E. (2005) *Data Mining: Practical Machine Learning Tools and Techniques*. Morgan Kaufmann.

WITTGENSTEIN, L. (1954) Philosophical investigations. *Phil. Rev.* **LXIII**: 530–559.

WOLOSZYN, M., STAUFFER, D., and KULAKOWSKI, K. (2007) Order–disorder phase transition in a cliquey social network. *Eur. Phys. J. B* **57**: 331 335.

WU, X., and LEVY, W. (2005) Increasing CS and US longevity increases the learnable trace interval. *Neurocomputing* **65–66**: 283–289.

XU, K., and LI, W. (2000) Exact phase transitions in random constraint satisfaction problems. *J. Artificial Intelligence Res.* **12**: 93–103.

XU, K., and LI, W. (2006) Many hard examples in exact phase transitions. *Theor. Computer Sci.* **355**: 291–302.

YULE, G. (1925) A mathematical theory of evolution, based on the conclusions of Dr. J. C. Willis, F.R.S. *Phil. Trans. Roy. Soc. London B* **213**: 221–87.

ZHOU, H. (2010) Solution space heterogeneity of the random k-satisfiability problem: theory and simulations. In *Proc. Int. Workshop on Statistical Mechanical Informatics*, Tokyo.

ZHOU, H., and MA, H. (2009) Communities of solutions in single solution clusters of a random k-satisfiability formula. *Phys. Rev. E* **80**: 066 108.

ZWEIG, K., PALLA, G., and VICSEK, T. (2010) What makes a phase transition? Analysis of the random satisfiability problem. *Physica A: Statistical Mechanics and its Applications* **389**: 1501–1511.

Index