Generalization Behaviour of Alkemic Decision Trees

K.S. Ng

Computer Sciences Laboratory
Research School of Information Sciences and Engineering
The Australian National University
kee@cslab.anu.edu.au

Abstract. This paper is concerned with generalization issues for a decision tree learner for structured data called ALKEMY. Motivated by error bounds established in statistical learning theory, we study the VC dimensions of some predicate classes defined on sets and multisets – two data-modelling constructs used intensively in the knowledge representation formalism of ALKEMY – and from that obtain insights into the (worst-case) generalization behaviour of the learner. The VC dimension results and the techniques used to derive them may be of wider independent interest.

1 Introduction

This paper is concerned with gaining some understanding of the generalization behaviour of ALKEMY, a logical decision-tree learner for structured data introduced under the higher-order logic learning framework of [17]. A brief (early) description of the learner appears in [6].

To get started on our goal, we turn to the rich body of literature on generalization issues. Inspection of error bounds established in statistical learning theory for general decision trees with arbitrary input domains and arbitrary node functions reveals that an important parameter governing the generalization behaviour of Alkemic decision trees is the VC dimension of node functions, and this is what we study in this paper. Specifically, we concentrate on some natural predicate classes defined on sets and multisets – two data-modelling constructs used intensively in the knowledge representation framework of [17] – and give bounds on their VC dimensions. The results turn out to have wider application beyond sets and multisets. Some indications of how they can be used to analyse common predicate classes defined on more complex data types like lists, trees, graphs, etc are given.

To the author's best knowledge, this is the first time that the VC dimensions of different predicate classes defined on sets and multisets have been analyzed *directly*. The only other relevant work I'm aware of is in [8], where the VC dimension of a class of predicates defined on sets is analyzed *indirectly* through a mapping to Blum's infinite attribute space model [5].

The paper is organized as follows. Section 2 provides background information on ALKEMY. Error bounds suitable for use with it are stated in Section 3. Sections 4 and 5 present the new VC dimension results. A discussion of the main findings is given in Section 6. We conclude in Section 7.

2 Alkemy

We assume some familiarity with a functional programming language like Haskell [26] in the following.

Figure 1 gives a high-level view of the ALKEMY classification learning system. It accepts as inputs (1) a set of training examples and (2) a hypothesis space, and produces as output a logical decision tree. A variant of the standard TDIDT algorithm is used to construct the output tree.

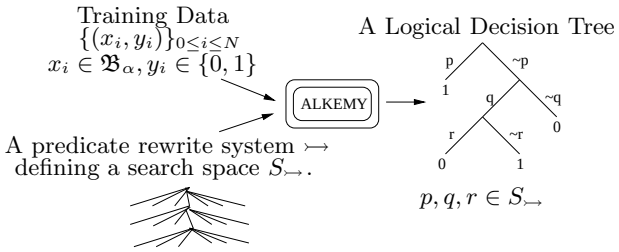

Fig. 1. A schematic diagram of Alkemy

Individuals (also known as instances) in the training set are represented using basic terms in the set $\mathfrak{B}_\alpha$ for some type α chosen appropriately according to the application. The formal basis for basic terms is provided in [17]; syntactically, they read like Haskell data constructs. A rich catalogue of data types is provided for data modelling via basic terms, and these include integers, floating-point numbers, characters, strings, booleans, data constructors, tuples, sets, multisets, lists, trees, graphs and composite types that can be built up from these.

Node functions are specified using predicate rewrite systems in ALKEMY. A detailed description of the mechanism is beyond the scope of this paper. Here we only provide sufficient detail in order to understand its use in Section 5.

Predicates are constructed incrementally by composing more basic functions called transformations. Composition is handled by the (reverse) composition function

$$\circ : (a \to b) \to (b \to c) \to (a \to c)$$

defined by $((f \circ g)\ x) = (g\ (f\ x))$.

Definition 1. *A* transformation *f is a function having a signature of the form*

$$f : (\varrho_1 \to \Omega) \to \cdots \to (\varrho_k \to \Omega) \to \mu \to \sigma,$$

where any type variables in $\varrho_1, \ldots, \varrho_k$ and σ appear in μ, and $k \geq 0$. (Here Ω is the type of the booleans.) The type μ is called the source *of the transformation, while the type σ is called the* target *of the transformation. The number k is called the* rank *of the transformation.*

The two constants 1 and 0 have type Ω. We now look at some examples of transformations.

Example 2. The transformation $\wedge_n : (a \to \Omega) \to \cdots \to (a \to \Omega) \to a \to \Omega$ defined by

$$\wedge_n\, p_1 \ldots p_n = \lambda x.((p_1\ x) \wedge \cdots \wedge (p_n\ x)),$$

provides a conjunction of n predicates.

Example 3. Each projection $proj_i : a_1 \times \cdots \times a_n \to a_i$ defined by

$$proj_i\ (t_1, \ldots, t_n) = t_i,$$

for $i = 1, \ldots, n$, is a transformation of rank 0.

Example 4. There are two fundamental transformations $top : a \to \Omega$ and $bottom : a \to \Omega$ defined by $(top\ x) = 1$ and $(bottom\ x) = 0$, for each x. The transformation *top* is the weakest predicate on the type a and *bottom*, the strongest.

Example 5. Let μ be a type and suppose $A, B, C : \mu$ are constants of type μ. Then, corresponding to A, one can define a transformation $(= A) : \mu \to \Omega$ by

$$((= A)\ x) = x = A,$$

with analogous definitions for $(= B)$ and $(= C)$. Similarly, one can define the transformations $(\neq A), (\neq B)$ and $(\neq C)$.

Example 6. Consider a type such as *Nat* (the type of the natural numbers) which has various order relations defined on it. Then, for any natural number N, one can define the transformation $(< N) : Nat \to \Omega$ by

$$((< N)\ m) = m < N.$$

In a similar way, one can define the transformations $(> N)$, $(\geq N)$, and $(\leq N)$.

Example 7. Consider the transformation $domCard : (\mu \to \Omega) \to \{\mu\} \to Nat$ defined by

$$domCard\ b\ t = card\ \{x \mid (b\ x) \wedge x \in t\},$$

where *card* computes the cardinality of a set. Given a predicate b on type μ and a transformation on *Nat* such as (> 42), one can construct a predicate $(domCard\ b) \circ (> 42)$ on sets of type $\{\mu\}$ which selects the subset of elements that satisfy the predicate b and then checks that the cardinality of this subset is greater than 42.

One can similarly define *domMcard* for multisets.

Example 8. Consider the transformation $setExists_1 : (a \to \Omega) \to \{a\} \to \Omega$ defined by

$$setExists_1\ b\ t = \exists x.((b\ x) \wedge (x \in t)).$$

The predicate $(setExists_1\ b)$ checks whether a set has an element that satisfies b.

Transformations are used to define a particular class of predicates, called standard predicates.

Definition 9. *A* standard predicate *is a term of the form*

$$(f_1\ p_{1,1} \ldots p_{1,k_1}) \circ \cdots \circ (f_n\ p_{n,1} \ldots p_{n,k_n}),$$

where f_i is a transformation of rank k_i ($i = 1, \ldots, n$), the target of f_n is Ω, p_{i,j_i} is a standard predicate ($i = 1, \ldots, n$, $j_i = 1, \ldots, k_i$), $k_i \geq 0$ ($i = 1, \ldots, n$) and $n \geq 1$.

Example 10. If p, q, and r are standard predicates (having appropriate type) and $\neg : \Omega \to \Omega$ is negation, then $(\wedge_3\ p\ q\ r) \circ \neg$ is a standard predicate.

Now we can very informally define a predicate rewrite system. A *predicate rewrite* is an expression of the form

$$p \rightarrowtail q,$$

where p and q are standard predicates. The predicate p is called the *head* and q is the *body* of the rewrite. A *predicate rewrite system* is a finite set of predicate rewrites. One should think of a predicate rewrite system as a kind of grammar for generating a particular class of predicates. Roughly speaking, this works as follows. Starting from the weakest predicate *top*, all predicate rewrites that have *top* (of the appropriate type) in the head are selected to make up child predicates that consist of the bodies of these predicate rewrites. Then, for each child predicate and each redex in that predicate, all child predicates are generated by replacing each redex by the body of the predicate rewrite whose head is identical to the redex. This generation of predicates continues to produce the predicate class. The space of predicates defined this way using a predicate rewrite system $\rightarrowtail$ is denoted $S_{\rightarrowtail}$.

Example 11. Consider the predicate rewrite system $\rightarrowtail$ given for the Musk problem in §5.1. The following is a path in the predicate space defined by $\rightarrowtail$.

$$top$$
$$setExists_1\ (\wedge_3\ top\ top\ top)$$
$$setExists_1\ (\wedge_3\ (proj_1 \circ (= -6))\ top\ top)$$
$$setExists_1\ (\wedge_3\ (proj_1 \circ (= -6))\ (proj_2 \circ (= 5))\ top)$$
$$setExists_1\ (\wedge_3\ (proj_1 \circ (= -6))\ (proj_2 \circ (= 5))\ (proj_{120} \circ (= 0)))$$

3 Error Bounds

We now state a few error bounds for general decision trees. The purpose is to determine what are important parameters that one should look at in analysing the generalization behaviour of Alkemic decision trees.

We start with a reminder of some basic concepts. In what follows, log denotes logarithm to base 2, ln denotes the natural logarithm, and $\lceil \cdot \rceil$ and $\lfloor \cdot \rfloor$ denote, respectively, the ceiling and floor functions. The set of natural numbers $\{1, 2, 3, \ldots\}$ is denoted $\mathbb{N}$.

Let X be an arbitrary set and $\mathcal{F}$ a class of predicates over X. The growth function of $\mathcal{F}$, $\Pi_{\mathcal{F}} : \mathbb{N} \to \mathbb{N}$, is defined by $\Pi_{\mathcal{F}}(n) = \max\{|\mathcal{F}_{|x}| : x \in X^n\}$, where

$$\mathcal{F}_{|x} = \{(f(x_1), \ldots, f(x_n)) : f \in \mathcal{F}\}.$$

Given $x \in X^n$, if $|\mathcal{F}_{|x}| = 2^n$, then we say x is *shattered* by $\mathcal{F}$. (Equivalently, we say a subset Y of X is shattered by $\mathcal{F}$ if each subset Z of Y can be picked out by a predicate in $\mathcal{F}$, i.e., there exists $f \in \mathcal{F}$ such that $\forall z \in Z.\ f(z) = 1$ and $\forall z \in Y \setminus Z.\ f(z) = 0$.) The Vapnik-Chervonenkis (VC) dimension of $\mathcal{F}$ is defined by

$$VCD(\mathcal{F}) = \max\{n : \Pi_{\mathcal{F}}(n) = 2^n\}$$

or ∞ if no such maximum exists.

The following is a standard result we will need. More facts about VC dimension can be found in standard texts like [1].

Proposition 12. *Let $\mathcal{F}$ be a finite predicate class. Then $VCD(\mathcal{F}) \leq \lfloor \log |\mathcal{F}| \rfloor$.*

Proof. We need at least 2^d predicates to shatter a set of d elements. □

Error bounds for decision trees obtained from classical VC theory suggest that the amount of training data needed for learning should grow at least linearly with the size of the tree and the VC dimension of the node functions. See, for example, [1], [12] and [13]. More recent results give data-dependent bounds that are qualitatively different from those earlier results. For example, in [14], the authors show how decision trees with node functions in $\mathcal{U}$ can be represented as thresholded convex combinations of functions in $\mathcal{U}$, and from that establish error bounds for decision trees using margin-based error bounds for two-layer neural networks (see [1] and [23]). We state the main theorem here. For more details, the reader is referred to [14] and [19].

Theorem 13 ([14]). *For a fixed $\delta > 0$, there is a constant c that satisfies the following. Let $\mathcal{D}$ be a distribution on $X \times \{0, 1\}$. Consider the class of decision trees of depth up to k, with node functions in $\mathcal{U}$. With probability at least $1 - \delta$ over the training set S of size m, every decision tree T satisfies*

$$\mathbf{P}_{(x,y)\sim\mathcal{D}}[T(x) \neq y] \leq \mathbf{P}_{(x,y)\sim S}[T(x) \neq y] + c\left(\frac{N_{\mathit{eff}}\, VCD(\mathcal{U}) \log^2 m \log k}{m}\right)^{1/3}.$$

Here N_{eff} is a data-dependent quantity that measures the *effective* number of leave nodes in T, a number that can be significantly smaller than the actual number of leave nodes in T. See, for the exact definition, [14].

The classical theorems are suitable for use with small trees; Theorem 13 works better for large trees.

4 Tools for Calculating VC Dimensions

As shown in the last section, the VC dimension of node functions is an important parameter in the generalization behaviour of Alkemic decision trees. To understand the nature of learning with ALKEMY, we thus need to develop methods to calculate the VC dimensions of (more-or-less arbitrary) predicate classes definable using predicate rewrite systems. The problem seems difficult at first sight; in fact, it was listed as an open research question in [17, Exercise 6.6]. But recent progress has shown that solutions to some important aspects of the general problem are actually rather straightforward. These results are reported here.

We will first outline the development of some useful tools for analysing predicate classes defined on sets and multisets in this section. Armed with these, we will then proceed in Section 5 to calculate the VC dimensions of three illustrative predicate rewrite systems selected from [17]. We remark that the tools developed here have applications beyond ALKEMY, for example in the analysis of systems that learn from set-valued objects like [8].

In this section, for the most part, we will abstract away from predicate rewrite systems and just work on predicate classes defined on 'collections' of natural numbers. As we shall see, this is a useful simplification since there is a simple mapping between natural numbers and arbitrary finite sets that we can exploit.

4.1 Sets

We will start with the following basic observation.

Proposition 14. *Let $\mathcal{F}_\exists$ be the class of predicates $\mathcal{F}_\exists = \{f_{i,j} : i, j \in \mathbb{N}, j \geq i\}$ where each $f_{i,j} : 2^{\mathbb{N}} \to \{0, 1\}$ is defined by*

$$f_{i,j}(t) = \begin{cases} 1 & \text{if } \exists x \in t.\ i \leq x \leq j \\ 0 & \text{otherwise.} \end{cases}$$

Then $VCD(\mathcal{F}_\exists) = \infty$.

Proof. It suffices to show that the subset $\mathcal{F}'_\exists = \{f_{i,i} : i \in \mathbb{N}\}$ of $\mathcal{F}_\exists$ has infinite VC dimension. For each $n \in \mathbb{N}$, we can construct a set $\{X_1, X_2, \ldots, X_n\}$ that is shattered by $\mathcal{F}'_\exists$ as follows. Enumerate all the subsets of $N = \{1, 2, \ldots, n\}$, assigning them numbers from 1 to 2^n. For instance, when $n = 3$ we get

1	2	3	4	5	6	7	8	.
$\emptyset$	$\{1\}$	$\{2\}$	$\{3\}$	$\{1,2\}$	$\{1,3\}$	$\{2,3\}$	$\{1,2,3\}$	

Now define X_i to be the set of all numbers assigned to a subset of N having i as a member. Continuing with our example for $n = 3$, we obtain the set

$$\{X_1 = \{2, 5, 6, 8\},\ X_2 = \{3, 5, 7, 8\},\ X_3 = \{4, 6, 7, 8\}\}.$$

It is clear that $\{X_1, X_2, \ldots, X_n\}$ constructed this way is shattered by $\mathcal{F}'_{\exists}$. □

We next give a useful generalization of Proposition 14. First, a definition.

Definition 15. *Let X be a set and $\mathcal{F}$ a class of predicates over X. We say a set $S \subseteq X$ is* disintegrated *by $\mathcal{F}$ if for every $x \in S$, there exists an $f \in \mathcal{F}$ such that $f(x) = 1$ and $f(y) = 0$ for all $y \in S \setminus \{x\}$.*

Lemma 16. *Let X be a set and suppose $\mathcal{F}$ is a class of predicates over X. Let $\mathcal{G} = \{g_f : f \in \mathcal{F}\}$ be the class of predicates where each $g_f : 2^X \to \{0, 1\}$ is defined by*

$$g_f(t) = \begin{cases} 1 & \text{if } \exists x \in t.\ f(x) = 1 \\ 0 & \text{otherwise.} \end{cases}$$

If there exists a finite $S \subseteq X$ such that $|S| \geqslant 2$ and S is disintegrated by $\mathcal{F}$, then $VCD(\mathcal{G}) \geq \lfloor \log |S| \rfloor$.

Proof. Proceeding as in Proposition 14, we can assign a different element of S to each subset of $N = \{1, 2, \ldots, \lfloor \log |S| \rfloor\}$. Defining X_i to be the set of all elements assigned to a subset of N in which i occurs gives us a subset of 2^X that is shattered by $\mathcal{G}$. □

As a simple application of Lemma 16, we give this next result for sets of tuples of constants.

Theorem 17. *Let N be a finite subset of $\mathbb{N}$ satisfying $|N| \geqslant 2$. Suppose $m \geq n > 0$ and let $\mathcal{G}_{m,n}$ be the class of predicates*

$$\mathcal{G}_{m,n} = \{g_{\{(i_l, j_l)\}_{1 \leq l \leq k}} : k \in \{1, \ldots, n\}, i_l \in \{1, \ldots, m\}, j_l \in N\}$$

where each $g_{\{(i_l, j_l)\}_{1 \leq l \leq k}} : 2^{N^m} \to \{0, 1\}$ is defined by

$$g_{\{(i_l, j_l)\}_{1 \leq l \leq k}}(t) = \begin{cases} 1 & \text{if } \exists (x_1, \ldots, x_m) \in t.(x_{i_1} = j_1) \wedge \cdots \wedge (x_{i_k} = j_k) \\ 0 & \text{otherwise.} \end{cases}$$

Then

$$VCD(\mathcal{G}_{m,n}) \geq \begin{cases} \lfloor m \log |N| \rfloor & \text{if } n = m; \\ \lfloor \log(\sum_{k=1}^{n} \binom{m}{k} (|N| - 1)^k) \rfloor & \text{otherwise.} \end{cases}$$

Proof. Let

$$\mathcal{F}_{m,n} = \{f_{\{(i_l,j_l)\}_{1\leq l\leq k}} : k \in \{1,\ldots,n\}, i_l \in \{1,\ldots,m\}, j_l \in N, i_1 < \cdots < i_k\}$$

where each $f_{\{(i_l,j_l)\}_{1\leq l\leq k}} : N^m \to \{0,1\}$ is defined by

$$f_{\{(i_l,j_l)\}_{1\leq l\leq k}}(x_1,\ldots,x_m) = \begin{cases} 1 & \text{if } (x_{i_1} = j_1) \wedge \cdots \wedge (x_{i_k} = j_k) \\ 0 & \text{otherwise.} \end{cases}$$

We use Lemma 16 to get the lower bounds here. When $n = m$, we can use for S the whole set N^m, which is clearly disintegrated by $\mathcal{F}_{m,m}$. When $n < m$, we construct S as follows. Pick an $x \in N$ at random and consider the following subset of $\mathcal{F}_{m,n}$:

$$\mathcal{F}_{m,n,x} = \{f_{\{i_l,j_l\}_{1\leq l\leq k}} : k \in \{1,\ldots,n\}, \\ i_l \in \{1,\ldots,m\}, j_l \in N \setminus \{x\}, i_1 < \cdots < i_k\}.$$

For each predicate $f_{\{(i_l,j_l)\}_{1\leq l\leq k}} \in \mathcal{F}_{m,n,x}$ add to S the tuple that has value j_l at the i_l-th component, and x everywhere else. (For instance, when $m = 5, k = 2, N = \{1,2,3\}$ and $x = 3$, given $f_{\{(1,2),(3,1)\}}$, we add $(3,3,1,3,2)$ to S.) It is not hard to see that each element in S can be picked out by the predicate that generated it. Further,

$$|S| = |\mathcal{F}_{m,n,x}| = \sum_{k=1}^{n} \binom{m}{k} (|N|-1)^k.$$

The condition $|N| \geqslant 2$ ensures that $|S| \geqslant 2$ in both cases. □

It is perhaps of (independent) interest to note that the dual $\mathcal{F}_\forall$ (defined below) of $\mathcal{F}_\exists$ defined in Proposition 14 has finite VC dimension.

Proposition 18. *Let $\mathcal{F}_\forall$ be the class of predicates $\mathcal{F}_\forall = \{f_{i,j} : i,j \in \mathbb{N}, j \geq i\}$ where each $f_{i,j} : 2^{\mathbb{N}} \to \{0,1\}$ is defined by*

$$f_{i,j}(t) = \begin{cases} 1 & \textit{if } \forall x \in t.\ i \leq x \leq j; \\ 0 & \textit{otherwise.} \end{cases}$$

Then $VCD(\mathcal{F}_\forall) = 2$.

Proof. It is easy to show that $VCD(\mathcal{F}_\forall) \geqslant 2$. Assume there exists a set $S = \{X, Y, Z\}$ that is shattered by $\mathcal{F}_\forall$. Clearly, none of the elements in S can be the empty set, which evaluates to 1 for each $f \in \mathcal{F}_\forall$. Further, each element in S must be finite. (Shattering is impossible otherwise.) Denote by $\max(A)$ and $\min(A)$ the biggest and smallest numbers in a (finite) set A of numbers and define the range of A by $range(A) = \{\min(A),\ldots,\max(A)\}$. We have

$$\forall A, B \in S,\ A \neq B \Rightarrow range(A) \nsubseteq range(B)$$

since if $range(A) \subseteq range(B)$, there is no way to make B true without also making A true. Without loss of generality, assume $\min(X) < \min(Y) < \min(Z)$. This implies $\max(X) < \max(Y) < \max(Z)$. Now, there is no $f_{i,j} \in \mathcal{F}_\forall$ such that $f_{i,j}(X) = 1$, $f_{i,j}(Z) = 1$, and $f_{i,j}(Y) = 0$ since any (i,j)-interval that covers both $\min(X)$ and $\max(Z)$ must also cover every number in the range $\{\min(Y), \ldots, \max(Y)\} \supseteq Y$. □

4.2 Multisets

We next look at multisets. The difference between a set and a multiset is that an element can occur multiple times in a multiset. Some of the results given for sets clearly carry over to multisets with little change. The multiplicity of elements allowed in multisets can sometimes be exploited, as done in our next result. First some notation.

Let A be a multiset of elements from some set X. In the following, we denote by $\#(A, x)$ the multiplicity of $x \in X$ in A. Further, we denote by $\mathbb{N}_0$ the set $\{0\} \cup \mathbb{N}$.

Definition 19. *Let A and B be multisets of elements from some set X. We define the pairwise maximum between A and B, denoted $A \sqcup B$, as follows: $A \sqcup B$ is the multiset that contains, for all $x \in X$, $\max\{\#(A,x), \#(B,x)\}$ occurrences of x. For example, $\{1,1,2,2,2\} \sqcup \{1,2,2,2,2,3,3,3\} = \{1,1,2,2,2,2,3,3,3\}$.*

Theorem 20. *Suppose X and Y are non-empty finite subsets of $\mathbb{N}$. Let $\mathcal{F}$ be the class of predicates $\mathcal{F} = \{f_{i,j} : i \in X, j \in Y\}$ where each $f_{i,j} : {\mathbb{N}_0}^{\mathbb{N}} \to \{0,1\}$ is defined by*

$$f_{i,j}(t) = \begin{cases} 1 & \text{if } \#(t,i) \geqslant j; \\ 0 & \text{otherwise.} \end{cases}$$

Let $d \in \mathbb{N}$. If $|Y| \geqslant d+1$ and $|X| \geqslant \binom{d}{i}$ for all $i \in \{1, \ldots, d\}$, then $VCD(\mathcal{F}) \geqslant d$.

Proof. The proof is in two stages. In the first stage, we show that given a function ψ from the powerset of $D = \{1, \ldots, d\}$ to ${\mathbb{N}_0}^{\mathbb{N}}$ satisfying a certain property, we can construct a set $Z = \{Z_1, \ldots, Z_d\}$ that is shattered by $\mathcal{F}$. In the second stage, we show that ψ exists and give a simple algorithm for constructing it.

Stage 1 We denote by (x, y) the multiset that contains y occurrences of x and nothing else. Assume ψ satisfies the following property: For all $S \subseteq D$, we have

1. $\psi(S) = (x, y)$ for some $x, y \in \mathbb{N}$, and
2. for all $A \subseteq D$ not equal to S, if $\psi(A) = (x, z)$ for some $z \in \mathbb{N}$ and $|A| \geq |S|$, then $S \subset A$ and $y > z$.

Given such a function ψ, define $Z_i = \bigsqcup \{\psi(S) : S \subseteq D, i \in S\}$ for each $i \in D$. (Example 23 below gives an example of a function ψ defined on the subsets of $D = \{1,2,3,4\}$ that satisfy the property stated above. References there to the

Label algorithm should be ignored for now. Each pair A (B, C) in the example should be interpreted as $\psi(A) = (B, C)$. For example, $\psi(\emptyset) = (1, 5)$. Note also the way each Z_i is defined using ψ.) We now argue that the set $Z = \{Z_1, \ldots, Z_d\}$ so-constructed is shattered by $\mathcal{F}$. Specifically, we show that for all $S \subseteq D$, $f_{x,y}(Z_i) = 1$ if $i \in S$ and $f_{x,y}(Z_i) = 0$ otherwise, given that $\psi(S) = (x, y)$.

Consider an arbitrary $S \subseteq D$ with $\psi(S) = (x, y)$. If $i \in S$, by construction, Z_i contains at least y occurrences of x and $f_{x,y}(Z_i) = 1$. Consider now the case when $i \notin S$. If $\#(Z_i, x) = 0$, then $f_{x,y}(Z_i) = 0$ as desired. If $\#(Z_i, x) > 0$, then there exists $A \subseteq D$ such that $i \in A$ and $\psi(A) = (x, z)$ for some $z \in \mathbb{N}$. We can assume without loss of generality that A is the set with the largest z. If $|A| \geq |S|$, then by the property of ψ, we have $y > z$ and $S \subset A$, which implies $f_{x,y}(Z_i) = 0$. If $|A| < |S|$, then by the property of ψ, we have $z > y$ and $A \subset S$. (Simply substitute the set A for the variable S and the set S for the variable A in the statement of the property of ψ.) This case can't arise since $A \subset S$ and $i \in A$ together imply $i \in S$, contradicting $i \notin S$.

Stage 2 It suffices to show that one such ψ exists. We will give a more general result that shows that not only does ψ exists, we can actually find many instances of it efficiently using well-studied algorithms in graph theory.

Given X and Y both non-empty finite subsets of $\mathbb{N}$, we first use the **Label** algorithm given below to label the subsets of D. For each $S \subseteq D$, we then define $\psi(S)$ to be the label assigned to S. To get some intuition, we first give a high-level description of the labelling algorithm. Conceptually, we first lay out in a sequence the subsets of D in groups, starting from the empty set (group 0), followed by the 1-subsets (group 1), the 2-subsets (group 2), ..., and finally finishing at D (group $|D|$). (A subset with k elements in it is called a k-subset here.) The algorithm starts by labelling the largest group and then iteratively label the next two largest unlabelled groups until every subset of D has a label.

We now give the algorithm. The variables l, u and m are integers. In the algorithm, we denote by $Y[i]$ and $X[i]$ the i-th largest elements in Y and X. The condition $|X| \geq \binom{d}{i}$ for all i comes about because of Step 2. The condition $|Y| \geq d + 1$ comes from the fact that there are $d + 1$ groups of subsets of D. Example 23 below gives a concrete example of the labelling. It is instructive to work through the example at this stage.

Alg. **Label**

1. $l \leftarrow 1$; $u \leftarrow 1$; $m \leftarrow \min\{i : \forall j.\ \binom{d}{j} \leqslant \binom{d}{i}\}$;
2. Label the m-subsets of D with $(X[i], Y[\lceil d/2 + 1 \rceil])$ in increasing order of i.
3. If $m - l < 0$, goto Step 6;
4. $C \leftarrow$ the $(m - l + 1)$-subsets of D;
5. For each $(m - l)$-subset S of D
   (a) Pick an $L \in C$ with label $(x, Y[m])$ such that $S \subset L$ and label S with $(x, Y[m + 1])$;
   (b) $C \leftarrow C \setminus L$;

6. If $m + u > d$, terminate;
7. $C \leftarrow$ the $(m + u - 1)$-subsets of D;
8. For each $(m + u)$-subset S of D
   (a) Pick an $L \in C$ with label $(x, Y[m])$ such that $L \subset S$ and label S with $(x, Y[m-1])$;
   (b) $C \leftarrow C \setminus L$;
9. $l \leftarrow l + 1$; $u \leftarrow u + 1$; Goto Step 3;

By design, the function ψ constructed from a labelling obtained by **Label**, assuming it terminates, satisfies the condition stated earlier. We now show that the **Label** algorithm always terminate successfully. For that, we need to show that Steps 5(a) and 8(a) can always be performed for each S. We will show this for Step 5(a); the argument for Step 8(a) is similar. What we are trying to do is in fact to find a matching in a bipartite graph. The vertices of the graph consists of the $(m - l)$ and $(m - l + 1)$-subsets of D, with the $(m - l)$-subsets forming the first partition, and the $(m-l+1)$-subsets the second. There is an edge from an $(m - l)$-subset A to an $(m - l + 1)$-subset B iff $A \subset B$. By the choice of m, we have

$$\text{no. of } (m-l)\text{-subsets} = \binom{d}{m-l} \leqslant \binom{d}{m-l+1} = \text{no. of } (m-l+1)\text{-subsets.}$$

Thus we seek a matching of cardinality $\binom{d}{m-l}$.

To show that such a matching exists and can be found efficiently, we introduce a concept from graph theory.

Definition 21. *A vertex cover of a graph $G = (V, E)$ is a set $U \subseteq V$ such that every edge of G is incident with a vertex in U.*

We make use of the following known result. For a proof, see, for example, [9].

Theorem 22 (König 1931). *The maximum cardinality of a matching in a bipartite graph G is equal to the minimum cardinality of a vertex cover of G.*

The set of $(m - l)$-subsets with cardinality $\binom{d}{m-l}$ is clearly a vertex cover. A straightforward indirect argument shows that there is no smaller vertex cover. The existence of our desired matching then follows from Theorem 22. There are efficient network flow algorithms for finding (all) such matchings; see, for instance, [21, Chap. 10].

Finally, the labelling algorithm will always terminate at Step 6 by the choice of m in Step 1. □

Example 23. Suppose $X = \{1, \ldots, 6\}$ and $Y = \{1, \ldots, 5\}$. Let $\mathcal{F}$ be as defined in Theorem 20. To construct a set $Z = \{Z_1, Z_2, Z_3, Z_4\}$ that is shattered by $\mathcal{F}$, we first label the subsets of $D = \{1, 2, 3, 4\}$ according to the **Label** algorithm.

One acceptable labelling is the following.

$$\emptyset\,(1,5)$$
$$\{1\}\,(1,4),\ \{2\}\,(4,4),\ \{3\}\,(2,4),\ \{4\}\,(3,4)$$
$$\{1,2\}\,(1,3),\ \{1,3\}\,(2,3),\ \{1,4\}\,(3,3),\ \{2,3\}\,(4,3),\ \{2,4\}\,(5,3),\ \{3,4\}\,(6,3)$$
$$\{1,2,3\}\,(1,2),\ \{1,2,4\}\,(5,2),\ \{1,3,4\}\,(2,2),\ \{2,3,4\}\,(6,2)$$
$$\{1,2,3,4\}\,(1,1)$$

Based on the function ψ obtained from the labelling, we construct

$$\begin{aligned} Z = \{\ & Z_1 = \{1,1,1,1,2,2,2,3,3,3,5,5\},\\ & Z_2 = \{4,4,4,4,1,1,1,5,5,5,6,6\},\\ & Z_3 = \{2,2,2,2,4,4,4,6,6,6,1,1\},\\ & Z_4 = \{3,3,3,3,5,5,5,6,6,6,2,2,1\}\ \}. \end{aligned}$$

It can be easily verified that Z is shattered by $\mathcal{F}$. ◀

Observation 24. *It is possible to weaken the condition on $|Y|$ in Theorem 20 using a more scrupulous grouping of the subsets, especially for large values of d. We note here a simple way to weaken that to $|Y| \geqslant d-1$ by treating the empty set as part of the 1-subsets, and the whole set D as part of the $(d-1)$-subsets during labelling. Labelling is possible because the empty set, being a subset of every other set, is connected to all the 2-subsets; and the set D, being a superset of every other set, is connected to all the $(d-2)$-subsets.*

5 Some Illustrations

Building on results presented in the previous section, we now analyse three instructive examples of predicate rewrite systems taken from [17, Chap. 6]. For each illustration, we briefly introduce the problem and give details on (1) the way individuals are represented; and (2) the predicate rewrite system used. Readers can consult [17] for more information.

5.1 Musk

This first illustration is the Musk problem described in [10]. Briefly, the problem is to determine whether or not a molecule has a musk odour. Molecules generally have many different conformations and, presumably, only one conformation is responsible for the activity. Each conformation is a tuple of 166 floating-point numbers, where 162 of these represent the distance in angstroms from some origin in the conformation out along a radial line to the surface of the conformation and the other four numbers represent the position of a specific oxygen atom. For convenience, the floating-point numbers are discretized into 13 intervals, resulting in the following.

Representation of Individuals

$-6, -5, \ldots, 5, 6 : \mathit{Distance}$
$\mathit{Conformation} = \mathit{Distance} \times \cdots \times \mathit{Distance}$
$\mathit{Molecule} = \{\mathit{Conformation}\}$

Here the product type $\mathit{Distance} \times \cdots \times \mathit{Distance}$ contains 166 components. The function *musk* to be learned has signature $\mathit{musk} : \mathit{Molecule} \to \Omega$.

Predicate Rewrite System

$\mathit{top} \rightarrowtail \mathit{setExists}_1\ (\wedge_3\ \mathit{top}\ \mathit{top}\ \mathit{top})$
$\mathit{top} \rightarrowtail \mathit{proj}_i \circ (= j)$ where $i \in \{1, 2, \ldots, 166\}, j \in \{-6, -5, \ldots, 6\}$

Proposition 25. $\mathit{VCD}(S_{\rightarrowtail}) = 30$.

Proof. By Proposition 12, $\mathit{VCD}(S_{\rightarrowtail}) \leq \lfloor \log |S_{\rightarrowtail}| \rfloor = \lfloor \log 1,679,615,641 \rfloor = 30$. We have the lower bound

$$\mathit{VCD}(S_{\rightarrowtail}) \geq \lfloor \log(\sum_{k=1}^{3} \binom{166}{k} (12)^k) \rfloor = \lfloor \log 1,295,658,552 \rfloor = 30$$

by Theorem 17. □

5.2 Climate

Consider next the problem of deciding whether a climate in some country is pleasant or not. The climate is modelled by a multiset. Each item in a multiset is a term characterizing the main features of the weather during a day and the multiplicity of the item is the number of times during a year a day with those particular weather features occurs.

Representation of Individuals

$\mathit{Sunny}, \mathit{Overcast}, \mathit{Rain} : \mathit{Outlook}$
$\mathit{Hot}, \mathit{Mild}, \mathit{Cool} : \mathit{Temp}$
$\mathit{High}, \mathit{Normal}, \mathit{Low} : \mathit{Humidity}$
$\mathit{Strong}, \mathit{Medium}, \mathit{Weak} : \mathit{Wind}$
$\mathit{Weather} = \mathit{Outlook} \times \mathit{Temp} \times \mathit{Humidity} \times \mathit{Wind}$

A climate is modelled as a multiset $\mathit{Climate} = \mathit{Weather} \to \mathit{Nat}$ and the function *pleasant* to be learned has signature $\mathit{pleasant} : \mathit{Climate} \to \Omega$.

Predicate Rewrite System

$$
\begin{aligned}
&top \rightarrowtail (domMcard\ top) \circ (> 0);\\
&top \rightarrowtail \wedge_4\ (projOutlook \circ top)\ (projTemp \circ top)\\
&\qquad\qquad (projHumidity \circ top)\ (projWind \circ top);\\
&top \rightarrowtail (= Sunny);\ top \rightarrowtail (= Overcast);\ top \rightarrowtail (= Rain);\\
&top \rightarrowtail (= Hot);\ top \rightarrowtail (= Mild);\ top \rightarrowtail (= Cool);\\
&top \rightarrowtail (= High);\ top \rightarrowtail (= Low);\ top \rightarrowtail (= Normal);\\
&top \rightarrowtail (= Strong);\ top \rightarrowtail (= Medium);\ top \rightarrowtail (= Weak);\\
&(> i) \rightarrowtail (> i + 50)\ \text{ where }\ i \in \{0, 50, \ldots, 300\}.
\end{aligned}
$$

Proposition 26. $8 \leq VCD(S_{\rightarrowtail}) \leq 11$.

Proof. By Proposition 12, $VCD(S_{\rightarrowtail}) \leq \lfloor \log |S_{\rightarrowtail}| \rfloor = \lfloor \log 2057 \rfloor = 11$. We use Theorem 20 to establish the lower bound. All the tuples of type *Weather* can be numbered and form the set X, with $|X| = 81$. Each predicate in $S_{\rightarrowtail}$ of the form

$$
\begin{aligned}
&(domMcard\ (\wedge_4\ (projOutlook \circ (= A))\\
&\quad (projTemp \circ (= B))\ (projHumidity \circ (= C))\ (projWind \circ (= D)))) \circ (> j)
\end{aligned}
$$

is equivalent to some $f_{i,j+1}$ as defined in Theorem 20, where i is the labelling number of (A, B, C, D). There are 81 ways to instantiate the variables A, B, C and D. The variable j can take on values in the set

$$Y = \{1, 51, 101, 151, 201, 251, 301, 351\}.$$

The largest d satisfying $|Y| \geq d - 1$ and $|X| \geq \binom{d}{i}$ for all i is $d = 8$. □

5.3 Beyond Sets and Multisets

Results like Theorem 17 and Theorem 20 are actually more useful than they appear. A natural thing to do when learning from structured data is to check for existence of substructures common to individuals of the same class. For example, given a graph, it is common to pull out the set of all subgraphs of a certain size and check whether there exists one satisfying a certain property. Similarly for lists, trees and other complex data types. This means that transformations involving sets and multisets actually appear very often in predicate rewrite systems defined over a wide range of structured data, and these can be analysed using results presented in this paper. We remark that, in fact, *all* but one illustrations described in [17, Chap. 6], which cover many different data types in common use, can be analysed this way.

To illustrate the kind of reasoning involved, we give one final example, again taken from [17], involving lists. We consider the East-West challenge proposed by Michalski. Given trains and the directions they are traveling in, the task is to learn a rule that can differentiate between those heading east and those heading west.

The most natural type to model a train is a list. We first introduce the types *Direction*, *Shape*, *Length*, *Kind*, *Roof*, and *Object*.

$East, West : Direction$
$Rectangular, DoubleRectangular, UShaped, BucketShaped,$
$Hexagonal, Ellipsoidal : Shape$
$Long, Short : Length$
$Closed, Open : Kind$
$Flat, Jagged, Peaked, Curved, None : Roof$
$Circle, Hexagon, Square, Rectangle, LongRectangle, Triangle,$
$InvertedTriangle, Diamond, Null : Object.$

We also introduce the following type synonyms for convenience.

$NumWheels = Nat$
$NumObjects = Nat$
$Load = Object \times NumObjects$
$Car = Shape \times Length \times NumWheels \times Kind \times Roof \times Load$
$Train = List\ Car.$

The function *direction* to be learned has signature $direction : Train \to Direction$.

Before giving the predicate rewrite system, we first introduce a few transformations for lists. The transformation $listToSet : Train \to \{Car\}$ converts a list of carriages into a set of carriages. The transformation $(sublists\ N) : Train \to \{Train\}$ takes a list of carriages and returns the set of all sublists of size N. The transformation $(!!N) : Train \to Car$ takes a train and returns the N-th carriage in the train. The predicate rewrite system is as follows.

$top \rightarrowtail listToSet \circ (setExists_1\ (\wedge_2\ top\ top))$;
$top \rightarrowtail (sublists\ 2) \circ (setExists_1\ (\wedge_2\ ((!!0) \circ top)\ ((!!1) \circ top)))$;
$top \rightarrowtail projShape \circ top$; $top \rightarrowtail projLength \circ top$; $top \rightarrowtail projNumWheels \circ top$;
$top \rightarrowtail projKind \circ top$; $top \rightarrowtail projRoof \circ top$; $top \rightarrowtail projLoad \circ top$
$top \rightarrowtail projObject \circ top$; $top \rightarrowtail projNumObjects \circ top$;
$top \rightarrowtail (= A)$ where A a constant of type *Shape*;
$top \rightarrowtail (= B)$ where B a constant of type *Length*;
$top \rightarrowtail (= C)$ where C a constant of type *Kind*;
$top \rightarrowtail (= D)$ where D a constant of type *Roof*;
$top \rightarrowtail (= E)$ where E a constant of type *Object*;
$top \rightarrowtail (= 1)$; $top \rightarrowtail (= 2)$; $top \rightarrowtail (= 3)$.

Proposition 27. $7 \leq VCD(S_{\rightarrowtail}) \leq 11$.

Proof. Given $|S_{\rightarrowtail}| = 2073$, we have $VCD(S_{\rightarrowtail}) \leq \lfloor \log |S_{\rightarrowtail}| \rfloor = 11$ by Proposition 12. The lower bound can be established by analysing the predicates generated by the first rewrite. The reasoning proceeds in a similar fashion as in Theorem 17, but taking into account the fact that the components of *Car* have different ranges. An element from each component is reserved as a default value, in the same way an $x \in N$ is used in Theorem 17. From that, we get a set X of *Car* objects that can be used to construct a shatterable set D of sets of *Car* objects, where $|D| = \lfloor \log |X| \rfloor$ by Lemma 16. Clearly, one can recover a *Train* object from each element in D. A straightforward counting exercise yields $|X| = 230$, giving us the lower bound $\lfloor \log |X| \rfloor = 7$. □

6 Discussion

In the three examples presented in the previous section, an upper bound on the VC dimension is established by counting the size of the predicate class. A lower bound is then given via an explicit construction of a set of individuals that is shattered by the predicate class, making use of the rich structures available. Interestingly, the upper and lower bounds are never too far apart, and this holds true for all the other illustrations in [17] we analysed. Now one would expect that it is possible to do a lot better than a naïve counting of the predicate class; apparently not. What are we to make of these results?

It was shown in [25] (see also [1, Chap. 5]) that for a predicate class with high VC dimension, there exist distributions that will force the learning algorithm to require a large number of examples to obtain good generalization. This, together with the results presented in this paper, implies that, in general, the true errors of hypotheses in the rich predicate classes used by ALKEMY cannot be easily estimated from empirical data, and that, *in the worst case*, the number of training examples needed grows rather quickly with the size and complexity of the hypothesis language used. The problem is that if we do not make any assumption about the underlying distribution, then we must be prepared to accept the possibility that everything can conspire against the learner – the more structures we introduce into the representation of individuals and the hypothesis language, the more structures there are to be exploited for producing bad cases.

7 Conclusion

We have looked at some generalization issues in relation to ALKEMY in this paper. In particular, we investigated the VC dimensions of some predicate classes defined on sets and multisets and studied their applications in the context of ALKEMY. The results provide valuable information on the nature of learning with sets and multisets, thus filling a gap in our understanding of the process of learning from structured data. On the practical side, the tools developed in this paper can be used to calculate the complexity of different predicate classes. In real applications, such calculations can be used to guide the selection and crafting of hypothesis languages.

Future Work We have shown in this paper that some fairly natural predicate classes defined on sets and multisets have high VC dimension. This implies that these classes are hard to learn in the distribution-free setting. However, learning with predicate classes that have high VC dimensions is possible if the underlying distribution is benign, and this information can be obtained from the training data. For instance, [24] shows that the VC dimension of a predicate class on the training sample can be used as a measure of how helpful the distribution is in identifying the target concept, and gives error bounds in terms of that. More recently, [4] gives error bounds in terms of the Rademacher and Gaussian complexities of predicate classes, and these can be estimated easily from the training data. PAC-Bayes and PAC-MDL bounds, which are also data-dependent results, can also help us obtain tighter bounds. Some relevant work along this line of research include [18] and [22]. Investigation into such data-dependent analysis is our future work.

Related Work A body of work in ILP has provided both upper and lower bounds on the number of examples required for learnability, mostly in the PAC setting. Upper bounds are usually obtained by analyzing concrete algorithms for learning restricted first-order classes; see, for example, [20], [11], [7] and [16]. Issues of computation and estimation, in the sense expounded in [1, §1.1], are usually tightly integrated in this kind of analyses, and this failure to separate concerns is slightly unsatisfactory.

Lower bounds, however, are usually obtained, independently of computation issues, using purely information-theoretic concepts like Vapnik-Chervonenkis dimensions. Examples of such work include [3], [15] and [2], and this paper is related to these. The fact that the same general conclusion was obtained from the analyses of two very different knowledge representation formalisms tells us something about the sample complexity of learning with rich expressive languages in general.

Acknowledgments

I'm grateful to John W. Lloyd and Evan Greensmith for valuable discussions.

References

1. Martin Anthony and Peter L. Bartlett. *Neural Network Learning: Theoretical Foundations*. Cambridge University Press, 1999.
2. Marta Arias and Roni Khardon. Complexity parameters of first order classes. In *Proceedings of the 13th International Conference on Inductive Logic Programming*, pages 22–37, 2003.
3. Hiroki Arimura. Learning acyclic first-order horn sentences from entailment. In *Proceedings of the International Conference on Algorithmic Learning Theory*. Springer-Verlag, 1997.
4. Peter L. Bartlett and Shahar Mendelson. Rademacher and Gaussian complexities: risk bounds and structural results. *Journal of Machine Learning Research*, 3:463–482, 2002.

5. Avrim Blum. Learning boolean functions in an infinite attribute space. *Machine Learning*, 9(4):373–386, 1992.
6. Antony F. Bowers, Christophe Giraud-Carrier, and John W. Lloyd. Classification of individuals with complex structure. In *Proceedings of the 17th International Conference on Machine Learning*, pages 81–88. Morgan Kaufmann, 2000.
7. William W. Cohen. PAC-learning recursive logic programs: Efficient algorithms. *Journal of Artificial Intelligence Research*, 2:501–539, 1995.
8. William W. Cohen. Learning trees and rules with set-valued features. In *Proceedings of the 13th National Conference on Artificial Intelligence*, pages 709–716, Menlo Park, CA, 1996. AAAI Press.
9. Reinhard Diestel. *Graph Theory*. Springer-Verlag, 2nd edition, 2000.
10. Thomas G. Dietterich, Richard H. Lathrop, and Tomás Lozano-Pérez. Solving the multiple instance problem with axis-parallel rectangles. *Artificial Intelligence*, 89:31–71, 1997.
11. Sašo Džeroski, Stephen Muggleton, and Stuart Russell. PAC-learnability of determinate logic programs. In *Proceedings of the Workshop on Computational Learning Theory*, 1992.
12. Andrzej Ehrenfeucht and David Haussler. Learning decision trees from random examples. *Information and Computation*, 82:231–246, 1989.
13. Usama Fayyad and Keki Irani. What should be minimized in a decision tree? In *Proc. the 8th National Conference on Artificial Intelligence*, pages 749–754, 1990.
14. Mostefa Golea, Peter L. Bartlett, Wee Sun Lee, and Llew Mason. Generalization in decision trees and DNF: Does size matter? In *Advances in Neural Information Processing Systems 10*, pages 259–265, 1998.
15. Roni Khardon. Learning function free horn expressions. *Machine Learning*, 37:241–275, 1999.
16. Jörg-Uwe Kietz and Sašo Džeroski. Inductive logic programming and learnability. *SIGART Bulletin*, 5(1):22–32, 1994.
17. John W. Lloyd. *Logic for Learning: Learning Comprehensible Theories from Structured Data*. Cognitive Technologies. Springer, 2003.
18. Yishay Mansour and David McAllester. Generalization bounds for decision trees. In *Proceedings of the 13th Annual Conference on Computational Learning Theory*, pages 69–80. Morgan Kaufmann, San Francisco, 2000.
19. Llew Mason. *Margins and Combined Classifiers*. PhD thesis, Research School of Information Sciences and Engineering, The Australian National University, 1999.
20. Stephen Muggleton and Cao Feng. Efficient induction of logic programs. In S. Muggleton, editor, *Inductive Logic Programming*, pages 281–298. Academic Press, 1992.
21. Christos H. Papadimitriou and Kenneth Steiglitz. *Combinatorial Optimization: Algorithms and Complexity*. Dover Publications, 1998.
22. Ulrich Rückert and Stefan Kramer. Towards tight bounds for rule learning. In *Proceedings of the 21st International Conference on Machine Learning*, 2004.
23. Robert E. Schapire, Yoav Freund, Peter Bartlett, and Wee Sun Lee. Boosting the margin: A new explanation for the effectiveness of voting methods. *The Annals of Statistics*, 26 (5):1651–1686, 1998.
24. John Shawe-Taylor, Peter L. Bartlett, Robert Williamson, and Martin Anthony. Structural risk minimization over data-dependent hierarchies. *IEEE Transactions on Information Theory*, 44(5):1926–1940, 1998.
25. Hans-Ulrich Simon. General bounds on the number of examples needed for learning probabilistic concepts. *J. of Computer and System Sciences*, 52:239–254, 1996.
26. Simon Thompson. *Haskell - The Craft of Functional Programming*. Addison-Wesley, 2nd edition, 1999.