[Journal Name], [Volumn Number], 1–23 ([Volumn Year])
© [Volumn Year] Kluwer Academic Publishers, Boston. Manufactured in The Netherlands.

Complexity–Based Induction

DARRELL CONKLIN conklin@qucis.queensu.ca
Department of Computing and Information Science, Queen's University, Kingston, Ontario, Canada, K7L 3N6

IAN H. WITTEN ihw@waikato.ac.nz
Department of Computer Science, University of Waikato, Hamilton, New Zealand

Editor: J. R. Quinlan

Abstract. A central problem in inductive logic programming is theory evaluation. Without some sort of preference criterion, any two theories that explain a set of examples are equally acceptable. This paper presents a scheme for evaluating alternative inductive theories based on an objective preference criterion. It strives to extract maximal redundancy from examples, transforming structure into randomness. A major strength of the method is its application to learning problems where negative examples of concepts are scarce or unavailable. A new measure called *model complexity* is introduced, and its use is illustrated and compared with a *proof complexity* measure on relational learning tasks. The complementarity of model and proof complexity parallels that of model and proof–theoretic semantics. Model complexity, where applicable, seems to be an appropriate measure for evaluating inductive logic theories.

Keywords: Inductive logic programming, data compression, minimum description length principle, model complexity, learning from positive–only examples, theory preference criterion

1. Introduction

Induction involves the generalization of facts into rules. Since the facts are covered by induced rules, they are logically redundant and need not be retained after inductive inference. Thus induction is, by definition, a form of data compression. The aim of this paper is to describe and illustrate the principle of complexity–based induction in the context of inductive logic programming. Informally, the principle judges theories by the amount of information they need to reproduce a given set of positive examples of a concept. It strikes a delicate balance between the complexity and generality of a theory.

This paper is structured as follows. In this first section we give a formal description of inductive logic programming, and discuss the problem of overgeneralization given positive–only examples. Section 2 develops complexity–based induction in detail, and gives two measures that can be applied directly to logic programs. Application of the technique to two relational concept learning problems is presented in Section 3. The model complexity measure introduced in the paper is shown in Section 4 to be an instance of a generalized model compression principle for inductive logic programming.

1.1. Inductive logic programming

We briefly review the logical foundation of inductive logic programming, as presented in [15]. This exposition of induction uses logic programming terms [11, 25]. A *logic program* is a finite set of rules of the form

$$A \texttt{ :- } B_1, \ldots, B_n.$$

where $n \geq 0$, A is an atom, and $B_1, \ldots, B_n$ are literals. The atom A is called the *head* of the rule, the B_i together constitute the *body* of the rule. When $n = 0$, the rule is called a *fact* or a unit clause. In such a case, the implication symbol `:-` is dropped and assumed present.

A *signature* for a logic program comprises variable, constant, and predicate symbols with their arities. A *term* in a given signature is either a variable, a constant, or an expression $f(t_1, \ldots, t_n)$, where $t_1, \ldots, t_n$ are terms and f is an n–ary function symbol. An *atom* is an expression $p(t_1, \ldots, t_n)$, where $t_1, \ldots, t_n$ are terms, and p is an n–ary predicate symbol. An atom is *ground* if it does not contain any variables. A *literal* is an atom or the negation of an atom. The *Herbrand base* $B(T)$ for a logic program T is the set of all possible ground atoms formed using its signature. A *Herbrand interpretation* is a subset of the Herbrand base. A *ground instance* of a rule has all variables that occur in the rule replaced by terms. A Herbrand interpretation I is a *Herbrand model* for a logic program if for each ground instance of a rule $A \texttt{ :- } B_1, \ldots, B_n$ in the program, A is in I if $B_1, \ldots, B_n$ are in I. The intersection of all Herbrand models for a program T is called a *least Herbrand model*, and denoted $M(T)$. A ground atom A is a *logical consequence* of a program T iff $A \in M(T)$. Logic programs, based on Horn-clause logic, lack the ability to explicitly represent negative facts. The inference of negative literals is based on a nonmonotonic rule of inference such as the closed world assumption, or negation as failure [11]: from $T \nvdash A$ infer $T \vdash \texttt{not } A$.

Logical consequence is computed by deductive inference. Inductive inference, on the other hand, generates new hypotheses, and may expand the least Herbrand model of a theory.[1] Suppose B is a logic program, representing background knowledge, and E is a set of independent facts. Inductive inference generates a hypothesis H subject to the following requirements:

necessity $B \nvdash E$,

consistency $B \cup E \nvdash \neg H$,

coverage $B \cup H \vdash E$.

The necessity condition ensures that the background knowledge does not already entail the examples — if it did, there would be no need for an inductive hypothesis. The consistency condition asserts that the negation of the hypothesis is not entailed by the background knowledge and examples — if it were, the hypothesis would be inconsistent with what is already known. The final condition dictates that the background knowledge and hypothesis, taken together, entail the observations. Henceforth we shall refer to the combination of the background knowledge and the hypothesis as a *theory*, and say that the theory *covers* the examples.

There will generally be an indefinite number of inductive hypotheses that meet the above requirements. As such, inductive inference is a highly underconstrained problem. The requirements say nothing about which hypothesis to prefer. This is the role of a *theory preference criterion*, an extra–logical rule that prefers one hypothesis to another. This paper does not discuss particular theory formation and generalization methods: it is concerned with evaluating proposed theories.

1.2. Concept learning

We now describe concept learning, a basic inductive inference task. Concept learning constrains general induction substantially by allowing only ground facts as examples, and assuming that each example has the same predicate symbol. A concept is simply a relation between entities in the world. Concepts can be unary (e.g., the concept of a *bird*), or of higher order (e.g., the *daughter* relation). An n–ary concept C is a set of ground atoms with a common n–ary predicate symbol, say, c. The *observation language* O for C is the set of all atoms in the Herbrand base with c as a predicate symbol. A concept C is always a subset of the observation language O. An instance of C is simply an element of C. A non–instance of C is an element of $O - C$.

The goal of concept learning is to construct a compact, intensional, definition of the set C, in finite time, from examples. If a machine had an infinite amount of time, data, and space, it could simply conjoin all examples to its growing theory. Finiteness restrictions necessitate using inductive inference to achieve or approximate the goal.

1.3. Generality and simplicity

Inductive inference is closely related to the process of generalization. Here we give an informal discussion of theory generality; see [4] for a more detailed description.

A logic program T' is *more general than* a logic program T iff $M(T') \supseteq M(T)$. This relation induces a lattice structure of logic programs. The examples E and the background knowledge B are points on this lattice, and any upper bound of $E \cup B$ is a theory conforming to the requirements of induction. In particular, any set E of examples is covered by both the simplest and most general theory

$$\top \overset{\text{def}}{=} \{\mathsf{c}(X_1, \ldots, X_n)\} \cup B,$$

where $X_1, \ldots, X_n$ are distinct variables, and the most complex and least general theory

$$\bot \overset{\text{def}}{=} E \cup B.$$

There may be more than one generalization of the examples in the lattice, and a preference criterion is needed to choose between them. One pervasive preference criterion is Occam's principle of parsimony.

Occam's principle. *Entia non sunt multiplicanda praeter necessitatem*,

literally, "entities should not be multiplied beyond necessity". Simpler theories are to be preferred.

Although theory simplicity is an ancient and intuitive notion, Occam's principle must be interpreted carefully to yield a useful preference criterion for machine learning, as the next section will demonstrate.

1.4. Overgeneralization and positive–only examples

Inductive inference machines employing Occam's principle under standard interpretation will not succeed when given only positive examples, as they are liable to select overly general theories. Consider the theory $\top$; clearly it is simple, but it is probably inaccurate because it is maximally general and indiscriminately accepts any new observation.

Very few concept learning systems can learn from positive–only data. Most systems use counterexamples of the concept to avoid overgeneralization: they maintain a separate set of negative examples, and insist that an induced theory cover no element of the set. Without negative examples, they will all produce $\top$ as an inductive theory. We construct some examples of this phenomenon, drawn from well–known relational and non–relational concept learning systems.

Mitchell [13] characterizes concept learning as searching for generalizations that cover all positive and no negative examples. The candidate elimination algorithm retains two sets, G and S, that are, respectively, the most general and most specific theories that cover the examples. To satisfy Occam's principle, select the simplest theory from the set G of maximally general theories. The theory G is initialized to $\top$. If no negative examples are given, $\top$ will be the simplest theory.

Quinlan's ID3 [19] inductive learning algorithm learns decision trees from examples. The ID3 algorithm finds an attribute to test, and for every possible outcome of the test, removes examples which meet the test from the set of current examples, creates a new branch in the decision tree, and recursively invokes itself with the remaining examples. The recursion terminates when all current examples are in the same class. If given only positive examples, this termination criterion will immediately succeed with the theory $\top$.

Winston [27] describes a general–purpose procedure for learning descriptions of structured scenes from examples. The first positive example forms the initial theory. Subsequent positive and negative examples ("near misses") provoke theory generalization and specialization, respectively. If no negative examples are given, the machine may steadily march towards the generalization $\top$.

Shapiro's Model Inference System [23] learns a concept by starting with the most general theory $\top$, and searching through a specialization hierarchy of logic programs, directed by negative examples. Only negative examples can remove $\top$ from its perch. Quinlan's FOIL system [20] cannot learn anything without negative examples, which are either explicitly labelled, or implicit by a closed world assumption. In the latter case, any observation which is not given as a positive example is assumed to be a non–instance.

The MARVIN concept learning system [22] uses background knowledge to guide search through a generalization hierarchy of first–order theories. In contrast to Shapiro's system, the most specific theory $\perp$ forms the initial hypothesis. Generalization operators are incrementally applied to the theory. Although MARVIN does not use negative examples, at each stage of generalization an oracle must be consulted to confirm that overgeneralization has not occurred.

The concept learning systems described above do not behave gracefully when negative examples are not present. This is because $\top$ is a generalization of any set of positive examples. The insistence on negative examples is, in many domains, quite artificial [3]. They require careful preparation, and learning machines lose a large degree of autonomy. The next section describes and applies an interpretation of Occam's principle that works in learning settings that include neither an oracle nor negative examples. Negative examples may be present; the interpretation has no problem with this.

The abolition of negative examples, however, comes with a price. Fundamental theorems of Gold [9] and Angluin [1] show that most hypothesis spaces (i.e., sets of possible theories) are not identifiable in the limit from positive examples. That is, an inductive inference machine may never converge to an intended or target theory. Identification in the limit from positive examples is a very strong property of a hypothesis space. It is, nevertheless, somewhat unrealistic in practical inductive logic programming settings. There is not, for example, an endless supply of positive examples. Given a small set of examples, we are interested in selecting, among competing theories, those that are likely to be more accurate. Complexity–based induction offers an objective selection method.

2. Complexity–based induction

Complexity–based induction theory [24, 5] motivates a fresh interpretation of Occam's principle, based on the idea of data compression. The metaphor is one of *communication*: the set of (positive) examples must be transmitted across a noiseless channel. Any similarity that is detected among observations can be exploited by giving them a more compact coding. Instances of the concept should have short codes, and non–instances should have no code at all.

According to complexity–based induction, the best theory for a concept is defined to be the one that minimizes the number of bits required to communicate the examples. The examples are communicated in two steps:

1. Encode and transmit a theory T,

2. Encode the examples using the theory T, and transmit this encoding.

Denote the code for T by $D(T)$, and the encoding of the examples by $D(E|T)$. If the codes for theories and examples form a *prefix set* (no code is a prefix of any other), upon receiving the catenation $D(T) \cdot D(E|T)$ a receiver can exactly reconstruct E.

The complexity of T is the length (in bits) of the string $D(T)$; denote this by $L(T)$. The code length of the examples E with respect to the theory T is the length (in bits) of $D(E|T)$; denote this by $L(E|T)$. If T does not cover E, $L(E|T)$ is defined to be infinite. Complexity–based induction dictates that the best theory T for the data E minimizes the description length $L(T|E)$, where

$$L(T|E) \overset{\text{def}}{=} L(T) + L(E|T). \tag{1}$$

A theory T is preferable to a theory T' if $L(T|E) < L(T'|E)$. If the description length is greater than or equal to the code length of the raw examples, useful induction has not taken place. Otherwise, valid generalizations have been made, redundancy has been extracted from E, and the data has been compressed. Random data cannot be compressed.

This is the essence of the *minimum description length principle* [21]. Complexity–based induction allows the induction of a theory from positive data without separate hypothesis testing. It is related to the theory of Bayesian inference [6], which prescribes maximizing the posterior probability of a theory given examples. In Bayesian terms, complexity–based induction assigns a prior probability distribution to theories which decreases with theory complexity. Both Solomonoff and Chaitin [24, 5] attempt to eliminate the subjectivity associated with prior probabilities by representing theories using Turing machines. Kolmogorov complexity [10] is also based on this "universal" prior distribution. Each of these notions of complexity is undecidable. Hence this prior probability must be somewhat subjective. The complexity measure for logic programs given in Section 2.4 strives to match an intuitive notion of complexity, but the notion of theory simplicity is a thorny one that cannot be resolved here. Applied research in Bayesian inference, and the minimum description length principle, has for the most part been confined to propositional theories. This paper shows that it can be applied to restricted types of first–order theories.

The quantity $L(T)$ measures the syntactic complexity of a theory T, and will be discussed in Section 2.4. The quantity $L(E|T)$ measures the complexity of the examples when coded using T. In a probabilistic setting this is simply the log likelihood of the data with respect to the theory. However, there is some question about what $L(E|T)$ should be in a logical setting. In this section, we present two ways to encode examples with respect to a theory. The first is the proof complexity measure of Muggleton et al. [17]. The second is our model complexity measure. This presentation will also analyze the relative strengths and weaknesses of each measure. Section 3 grounds this analysis by applying the measures to relational learning tasks.

2.1. Proof complexity

The proof complexity measure ($\mathcal{PC}$) works by identifying the proofs of examples in E with respect to the theory T. In Muggleton et al. [17], this is the sequence of choice points in an SLD–refutation, using the standard Prolog leftmost computation rule [11]. The choice points in a refutation are easy to encode. Let `:-` $G_1, \ldots, G_n$ be the current goal, and suppose this goal is at the root of a success branch in the SLD–tree. Assuming a leftmost computation rule, suppose there are k rules where G_1 unifies with the rule head. It will require $\log_2 k$ bits to select one of these points in the SLD–tree.[2] This selection of choice points continues until a refutation is reached. The bits required at each choice point are accumulated and added together to give the final proof complexity measure. If we denote the code length of an atom A with respect to a theory T by $L_{\mathcal{PC}}(A|T)$, then[3]

$$L_{\mathcal{PC}}(E|T) \overset{\text{def}}{=} \sum_{A \in E} L_{\mathcal{PC}}(A|T). \tag{2}$$

For non–generative rules, that is, rules where the head contains one or more (non–generative) variables not occurring in the rule body, bindings for these variables must be specified. For example, given the rule `c(X)`, and the example `c(a)`, the substitution `X/a` must be specified. Here for simplicity we assume function–free logic programs, so that only a finite number of ground substitutions are possible. Then given a theory with c constants, each substitution to a non–generative variable will require $\log_2 c$ bits.

Muggleton [14] gives a variant of this method. Feldman [7] also uses a proof–theoretic measure of complexity, called *derivational complexity*, for evaluating inductive inference of context–free grammars. There is a similarity between proof complexity and explanation–based learning; adding redundant generalizations to a theory may speed up performance for similar future examples, but also may degrade overall performance. This is commonly known as the utility problem of explanation–based learning [12]. Proof complexity can offer a principled objective measure for choosing when to save a generalization.

2.2. Model complexity

The model complexity measure ($\mathcal{MC}$) works by measuring the size of a subset of the least Herbrand model. The least Herbrand model of a logic program T contains many atoms, and to transmit the examples E we can restrict attention to a subset of it, namely those atoms $Q(T)$ that are also possible observations:

$$Q(T) \overset{\text{def}}{=} O \cap M(T). \tag{3}$$

In other words, $Q(T)$ is the set of observable atoms covered by T. Borrowing Popper's term [18], we call $Q(T)$ the *empirical content* of a theory.

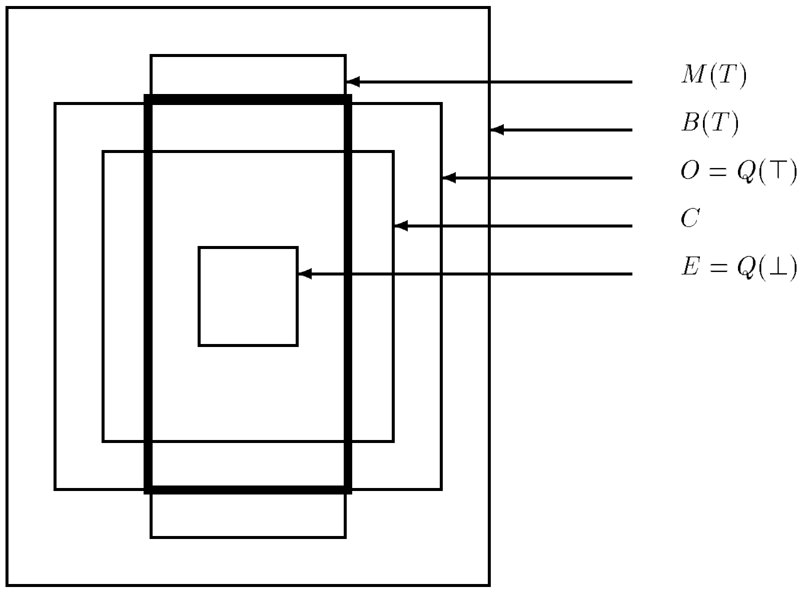

Figure 1. The topology of the models of induction. The thick boundary encloses the empirical content $Q(T)$ of a theory T. This is the smallest set containing the examples E which can be reconstructed by a receiver of T.

Figure 1 shows the topological relationships between the various models of induction, with the set $Q(T)$ highlighted. The goal of concept learning is to find a theory T that makes the set $Q(T)$ identical to the concept set C. At the outer level is the Herbrand base $B(T)$. The set $B(T) - O$ contains all atoms which are not observable. The set $O - C$ contains all non–instances of the concept C; as we have seen, existing inductive inference systems rely heavily on a presentation of this set in the form of oracle queries or explicit negative examples. Note that the least Herbrand model $M(T)$ of a theory T may contain non–observable atoms, and the empirical content $Q(T)$ may contain non–instances. The set $B(T) - M(T)$ contains all those atoms whose negation can be inferred using the closed world assumption. Quinlan's FOIL system [20], if not given explicit negative examples, uses this principle; it infers every atom in the set $O - E$ as an implicit negative example. This assumption can be stifling; consider the trivial concept $C = \{\mathtt{c(a)}, \mathtt{c(b)}\}$ and the example set $E = \{\mathtt{c(a)}\}$. The only theory considered under the closed world assumption is $\perp$, thus the admissible theory $\{\mathtt{c(X)}\}$ is rejected. On the other hand, without a similar assumption, concept learning systems suffer from the problems associated with overgeneralization and positive-only examples, as discussed in Section 1.4. Complexity–based concept learning systems can safely use an open world semantics, making no assumptions about the truth values of observable atoms not explicitly presented as positive or negative examples.

Inductive inference hypothesizes a theory T, and this theory will have an empirical content $Q(T)$. There are three possible relations between the set of examples E and $Q(T)$:

1. $E = Q(T)$. The theory covers all the examples and no other observable atoms.
2. $E \not\subseteq Q(T)$. There are examples not covered by the theory.
3. $E \subseteq Q(T)$. The theory covers all the examples, and perhaps other observable atoms.

In case 1, only the theory T need be transmitted, since the receiver can reconstruct E exactly by deductive inference. As E grows, however, it becomes increasingly unlikely that a simple theory covers E and only E. If case 2 holds, we must add the set $E - Q(T)$ — the exceptions to the theory — explicitly to T, thus ensuring that the theory conforms to the coverage requirement of induction. This situation is not depicted in Figure 1 because it is assumed that T has already been so augmented. In case 3, the common situation, the theory is perhaps more general than E — that is fine, to an extent. It indicates that induction has taken place. However, further information must be transmitted to convey the set E. This can be done by specifying the relevant subset $Q(T) - E$ of the model $Q(T)$. By sending a string of length

$$L_{\mathcal{MC}}(E|T) \overset{\text{def}}{=} \log_2 \binom{|Q(T)|}{|E|} \tag{4}$$

bits, a subset of $Q(T)$ of either size $|E|$ or size $|Q(T)| - |E|$ can be identified. One extra bit suffices to communicate which of these is meant. Intuitively, $L_{\mathcal{MC}}(E|T)$ is 0 when either $E = Q(T)$ (e.g., $T = \perp$) or $E = \emptyset$.

This model complexity measure requires that the observation language O is finite; to evaluate (4), the size of $Q(T)$ must be measured. Note however that $\mathcal{MC}$, like $\mathcal{PC}$, can handle background knowledge involving function symbols. Future research will concentrate on modifying the model complexity measure to apply to concepts involving function symbols.

2.3. *Discussion*

Let us look at the consequences of adopting one of these methods for encoding examples. In the $\mathcal{PC}$ measure, given two theories of equal syntactic complexity, the theory that most efficiently generates the examples will be preferred. The $\mathcal{MC}$ measure, on the other hand, will prefer the theory with the smaller empirical content. Each measure has strengths and weaknesses. $\mathcal{PC}$ can apply to domains where the observation language O is infinite, whereas $\mathcal{MC}$ requires that O is finite. The $\mathcal{MC}$ measure is semantic, dependent on the model of a theory, whereas $\mathcal{PC}$ is a purely syntactic preference criterion, dependent on a particular proof strategy. This points to a weakness of $\mathcal{PC}$; the SLD refutation performed by Prolog's left-most computation rule may not be the most economical one, according to the $\mathcal{PC}$ measure. Searching for the most economical proof for every example is out of the question due to intractability. More dramatically, Prolog's depth–first search rule is incomplete. Hence it is possible that $\mathcal{PC}$ will fail to produce a measure for a theory. Whereas $\mathcal{MC}$ also requires proofs to compute the model $Q(T)$, and is subject to the latter criticism, the former one does not apply; $\mathcal{MC}$ makes no distinction between two different refutations.

The $\mathcal{PC}$ measure requires that all examples be encoded and transmitted sequentially. This is a weakness in the method, as there will likely be much redundancy in their SLD-trees. The $\mathcal{MC}$ measure, on the other hand, does a "parallel" encoding of the examples, in the sense that it transmits an identification for a model rather than for a specific set of proofs. Since the examples are assumed to be drawn from a stationary distribution, it is unnecessary to preserve order information when communicating them. A related problem with the $\mathcal{PC}$ measure is that it will require some bits to communicate the examples even if the theory exactly covers them (case 1, Section 2.2). This is counterintuitive; if a theory covers only the examples, and that theory is transmitted, it should not be necessary to send further bits to identify the examples. For example, according to (2),

$$L_{\mathcal{PC}}(E|\perp) = \sum_{A \in E} L_{\mathcal{PC}}(A|\perp) = \sum_{A \in E} \log_2 |E| = |E| \times \log_2 |E|$$

bits, but according to (4), $L_{\mathcal{MC}}(E|\perp) = 0$ bits, as the empirical content of the theory $\perp$ is equal to the examples E (recall Expression 3).

program	$\rightarrow$	rule . program	(0.5)
program	$\rightarrow$	Λ	(0.5)
rule	$\rightarrow$	fact	(0.5)
rule	$\rightarrow$	head `:-` body	(0.5)
head	$\rightarrow$	atom	(1.0)
fact	$\rightarrow$	atom	(1.0)
body	$\rightarrow$	literal , body	(0.5)
body	$\rightarrow$	literal	(0.5)
literal	$\rightarrow$	atom	(0.5)
literal	$\rightarrow$	`not` atom	(0.5)
atom	$\rightarrow$	(see text)	

Figure 2. A probabilistic grammar for logic programs.

Despite the fundamental differences between these two measures, on another level they are similar, according to the following informal argument. Theories with smaller, or more specific, empirical content will tend to be more complex. Very general theories will tend to be simple. Also, complex theories will tend to have shorter proofs for the examples from which they were induced. Therefore, theories with smaller empirical content will tend to have shorter proofs.

2.4. *Coding theories*

We have discussed two methods for coding examples with respect to a logic program, and now turn to the problem of coding the theories themselves. For simplicity, and to facilitate comparison of the two complexity measures above, function–free logic programs are assumed.

The easiest way to code a logic program is to transmit it directly in, say, ASCII format. This, however, is extremely inefficient. Like examples, theories contain redundancy and can be compressed; there is redundancy inherent in the grammar of well–formed theories, and there is redundancy introduced by a particular use of the grammar.

There are numerous valid ways to code logic programs. We use the following simple, but reasonably efficient, scheme. Well–formed logic programs have an unambiguous probabilistic context–free grammar [26], shown in Figure 2. The numbers on the right are production probabilities. The probability $P(T)$ of a particular theory T with respect to the grammar is the product of the production probabilities used in its derivation.

The code length $L(T)$ of the theory is $-\log_2 P(T)$ bits. All that remains is to specify the probability of atoms. This can be done using the theory's signature — the variables, constants, and predicate symbols that appear in the program —

along with their arities. In general, if there are c constants, p predicate symbols and v variables in the signature, to transmit an atom of arity a will require

$$\log_2 p + a \log_2(v + c) \tag{5}$$

bits: $\log_2 p$ bits to identify the predicate symbol, and $\log_2(v + c)$ bits to specify an argument — a variable or a constant — for each of the a positions in the atom.

Whereas the ground atoms formed from the constants and predicate symbols (the Herbrand base) are fixed in advance, different theories may need a different number of symbols for variables. To accommodate this, we preface each theory with a code of length $\log_2(v + 1)$ bits, identifying the number of variables v (including the possibility that $v = 0$). The number of variables needed to express a theory is simply the maximum over the number of distinct variables in each rule.

The coding method can be paraphrased as follows. The number of bits required to code a logic program is the sum of

- $\log_2(v + 1)$ bits, where v is the number of variables in the program,
- 1 bit per program,
- 2 bits per rule in the program,
- 2 bits per literal in the body of each rule,
- bits for all atoms in the program.

Note that we do not bother to prefix the transmission of the theory with its lexicon. This is for simplicity of presentation, and also because the lexicon (aside from the number of variables and variable names) is assumed constant for all inductive logic programs. Since we are mainly interested in the relative quality of competing inductive logic programs leaving out the lexicon does not affect the result.

This coding method for logic programs is certainly not optimal, but it is simple and easy to understand. Some immediate ideas for improvement are to 1) base the probability of a literal on its relative frequency of occurrence in the theory, 2) remove redundant rules in a theory, or redundant literals in rules [4] before theory encoding, and 3) take into account the redundancy of clause orderings or literal orderings within the body of a clause. Exploring methods for efficiently coding logic programs is a worthwhile area for future research.

In Section 2.3, we stated that simple inductive theories *tend* to be more general; this statement can now be elaborated upon. Consider the two simple rules of structural generalization, one which adds a rule to a program, and another which deletes a literal from a rule:

1. For any theory T and rule r, $M(T) \subseteq M(T \cup \{r\})$.
2. For any theory T, clause c, and literal l, $M(T \cup \{c \vee l\}) \subseteq M(T \cup \{c\})$

Rule 2 does in fact decrease theory complexity according to the grammar of Figure 2, since $L(T \cup \{c\}) \leq L(T \cup \{c \vee l\})$. Rule 1, however, increases theory complexity. Simpler logic programs are not necessarily more general.

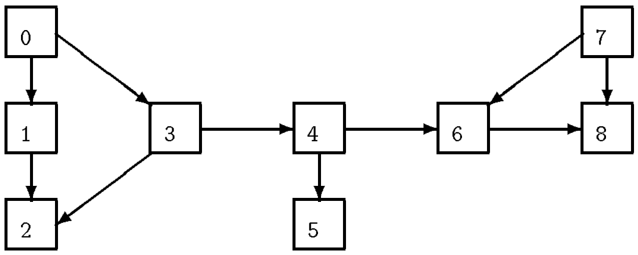

Figure 3. An example network illustrating the binary reachability relation.

3. Examples of complexity–based induction

A small Prolog metaprogram has been developed to compute syntactic theory complexity, and the $\mathcal{MC}$ and $\mathcal{PC}$ complexity measures. This section will present some initial results we have obtained in the application of complexity–based induction to two relational learning tasks.

3.1. *Networks*

This application of complexity–based induction is learning a general network relation from examples, originally discussed by Quinlan [20]. The concept under consideration is the binary "reachability" relation in a directed graph. One vertex can reach another if there is a path between them in the graph.

The signature comprises two binary predicates `reach` and `linked`, along with nine constants $\{\texttt{0}, \ldots, \texttt{8}\}$. The background knowledge contains an extensional definition of the predicate `linked` for a particular network (see Figure 3):

$$\begin{array}{ll} \{ & \langle \texttt{0},\texttt{1}\rangle.\ \langle \texttt{0},\texttt{3}\rangle.\ \langle \texttt{1},\texttt{2}\rangle.\ \langle \texttt{3},\texttt{2}\rangle.\ \langle \texttt{3},\texttt{4}\rangle. \\ & \langle \texttt{4},\texttt{5}\rangle.\ \langle \texttt{4},\texttt{6}\rangle.\ \langle \texttt{6},\texttt{8}\rangle.\ \langle \texttt{7},\texttt{6}\rangle.\ \langle \texttt{7},\texttt{8}\rangle\}. \end{array}$$

The notation $\langle x, y\rangle$ is shorthand for the fact $\texttt{linked}(x, y)$, meaning that there is a directed edge between vertices x and y in the network. Below we ignore the complexity of this background knowledge, as it is constant for all proposed theories.

The observation language O is the set of all atoms of the form $\texttt{reach}(x, y)$, where x, y are constants. The example set E is a complete specification of the predicate `reach` for the network in Figure 3:

$$\begin{array}{ll} \{ & \langle \texttt{0},\texttt{1}\rangle.\ \langle \texttt{0},\texttt{2}\rangle.\ \langle \texttt{0},\texttt{3}\rangle.\ \langle \texttt{0},\texttt{4}\rangle.\ \langle \texttt{0},\texttt{5}\rangle.\ \langle \texttt{0},\texttt{6}\rangle.\ \langle \texttt{0},\texttt{8}\rangle. \\ & \langle \texttt{1},\texttt{2}\rangle.\ \langle \texttt{3},\texttt{2}\rangle.\ \langle \texttt{3},\texttt{4}\rangle.\ \langle \texttt{3},\texttt{5}\rangle.\ \langle \texttt{3},\texttt{6}\rangle.\ \langle \texttt{3},\texttt{8}\rangle.\ \langle \texttt{4},\texttt{5}\rangle. \\ & \langle \texttt{4},\texttt{6}\rangle.\ \langle \texttt{4},\texttt{8}\rangle.\ \langle \texttt{6},\texttt{8}\rangle.\ \langle \texttt{7},\texttt{6}\rangle.\ \langle \texttt{7},\texttt{8}\rangle\}. \end{array}$$

Table 1. Theories for networks, and their complexities.

i	T_i	$L(T_i)$
1	`reach(X,Y).`	12.5

2	`reach(0,1). reach(0,2). reach(0,3). reach(0,4).`
	`reach(0,5). reach(0,6). reach(0,8). reach(1,2).`
	`reach(3,2). reach(3,4). reach(3,5). reach(3,6).`
	`reach(3,8). reach(4,5). reach(4,6). reach(4,8).`
	`reach(6,8). reach(7,6). reach(7,8).`
		178.5

3	`reach(X,Y) :- linked(X,Y).`
	`reach(0,2). reach(0,4). reach(0,5).`
	`reach(0,6). reach(0,8). reach(3,5).`
	`reach(3,6). reach(3,8). reach(4,8).`
		111.7

4	`reach(X,Y) :- linked(X,Y).`
	`reach(X,Y) :- linked(X,Z).`
		43.7

5	`reach(X,Y) :- linked(X,Y).`
	`reach(X,Y) :- linked(X,Z), linked(Z,Y).`
	`reach(0,5). reach(0,6).`
	`reach(0,8). reach(3,8).`
		94.5

6	`reach(X,Y) :- linked(X,Y).`
	`reach(X,Y) :- linked(X,Z), reach(Z,Y).`
		53.8

Table 2. Encoding 19 examples of *reach* using model complexity and proof complexity.

T	$L(T)$	$L_{\mathcal{MC}}(E\|T)$	$L_{\mathcal{MC}}(T\|E)$	$L_{\mathcal{PC}}(E\|T)$	$L_{\mathcal{PC}}(T\|E)$
$T_1 = \top$	12.5	60.4	72.9	120.5	133.0
$T_2 = \bot$	178.5	0	178.5	80.7	259.2
T_3	111.7	0	111.7	96.3	208.0
T_4	43.7	47.4	91.1	110.6	154.3
T_5	94.5	0	94.5	101.9	196.5
T_6	53.8	0	53.8	106.1	160.0

Table 1 presents six inductive theories that cover the examples. Theory T_1 is the most general theory $\top$. It has one rule with two non–generative variables `X` and `Y`. Theory T_2 is the least general theory $\bot$: it is simply the 19 examples expressed as facts. The first rule of theory T_3 states that two vertices are reachable if they are linked; since this rule alone is incomplete (i.e., does not cover all examples), it must be augmented by facts (case 2 in Section 2.2). Theory T_4 has an overly–general rule which states that vertices `X` and `Y` are reachable if `X` is linked to some vertex. This rule has a non–generative variable `Y`. The first rule of theory T_4 is logically redundant, given the second. The first two rules of theory T_5 cover all but 4 examples: hence it is augmented by these examples. Finally, theory T_6 concisely expresses the reachability relation using a recursive rule.

Table 1 also presents the code lengths of the theories, according to the theory complexity measure presented in Section 2.4. It is satisfying that the most general theory T_1 and the most specific theory T_2 are the least and most complex theories, respectively. Thus the prior probability on theories corresponds at least to some extent to our subjective notion of theory simplicity. Note also that the $\leq$ relation on theory complexity connects strongly with the $\supseteq$ relation on models: for the theories of Table 1 we have $L(T_1) \leq L(T_4) \leq L(T_6)$ and $Q(T_1) \supseteq Q(T_4) \supseteq Q(T_6)$.

Table 2 presents the evaluation of each theory according to the $\mathcal{MC}$ and $\mathcal{PC}$ measures. The ranking of theories given by each measure is as follows:

$\mathcal{MC}$	T_6	T_1	T_4	T_5	T_3	T_2
$\mathcal{PC}$	T_1	T_4	T_6	T_5	T_3	T_2

Both measures agree on the bottom three theories, in particular, theory T_2 or $\bot$ is given the lowest valuation. Each measure requires the fewest bits to code the examples using $\bot$ (0 for $\mathcal{MC}$, 80.7 for $\mathcal{PC}$), and the most bits to code the examples using $\top$ (60.4 for $\mathcal{MC}$, 120.5 for $\mathcal{PC}$). However, they differ in their ranking of the top three theories. The $\mathcal{MC}$ measure prefers the most intuitively satisfying theory T_6; its closest competitor is the theory T_1 or $\top$. The $\mathcal{MC}$ measure will continue to prefer T_6 even as the number of available positive examples is decreased to 16. The ranking of theories produced by $\mathcal{PC}$ is somewhat disappointing, since it gives the highest valuation to the most general theory T_1. It is of considerable concern that, for this task, the ranking produced by the $\mathcal{PC}$ measure is identical to one based only on theory simplicity. The "correct" theory T_6 is ranked third. This behavior is due to the fact discussed earlier that $\mathcal{PC}$ does not take proof correlations into account, and requires that the examples be transmitted in sequential fashion. The theory T_6, for example, produces some relatively deep proofs: the example $\langle \texttt{0}, \texttt{8} \rangle$ requires 9.3 bits to encode. The example $\langle \texttt{0}, \texttt{6} \rangle$ has a very similar proof tree, and the $\mathcal{PC}$ measure could benefit by taking this similarity into account. It seems that $\mathcal{MC}$ performs well in this relational domain of networks.

3.2. *Chess endgame*

The second application of complexity–based induction is learning the "illegality" relation in a chess endgame comprising a black king versus a white king and white rook. A position is an illegal white-to-move position if two or more pieces are placed on the same square, the kings are on adjacent squares, or the black king is in check. This example has been studied extensively in the inductive logic programming literature [16].

The signature comprises the 6–ary predicate `illegal`, the binary predicate `adj`, and 8 constants $\{\texttt{1}, \ldots, \texttt{8}\}$. The background knowledge contains a definition of the `adj` predicate:

```
adj(X,Y) :- X is Y - 1.
adj(X,Y) :- X is Y + 1.
```

Again we ignore the complexity of this background knowledge, as it contributes a fixed number of bits to the complexity of any inductive theory.

The observation language O is the set of all atoms of the form $\texttt{illegal}(a, b, c, d, e, f)$ where a through f are constants. This relational learning task is challenging for the $\mathcal{MC}$ measure because $|O| = 8^6 = 262144$, making it somewhat expensive to compute $|Q(T)|$ and evaluate Expression 4. In the network task (Section 3.1) $|Q(T)|$ can be computed following $|O| = 81$ invocations of Prolog's `call` predicate. For the chess endgame task such a procedure is unworkable in a practical sense. Instead we use standard statistical estimation techniques to estimate the proportion p of elements of O that are in the model $M(T)$. An estimate of $|Q(T)|$ can be expressed as

$$\overline{|Q(T)|} \stackrel{\text{def}}{=} \overline{p} \times |O| = \frac{s}{n} \times |O| \tag{6}$$

where $\overline{p}$ is the estimate of the proportion p, n is the number of randomly sampled observable atoms given to `call`, and s is the number of successes.[4] By setting n sufficiently high, $\overline{p}$ can have an arbitrarily small standard error. It can happen that $\overline{|Q(T)|}$ is less than the number of examples $|E|$. Since this is in contradiction of the fact that $\overline{|Q(T)|}$ must be at least as large as $|E|$ (from (3) and the requirement that $T \vdash E$), if it occurs we set $\overline{|Q(T)|}$ to $|E|$.

Table 3 presents 7 theories for the chess endgame task.[5] Theory T_7 is the 92% accurate theory discovered by FOIL after 100 examples using an explicit negative example approach. Theory complexities are not displayed in the table, as most theories must be augmented by examples in order to satisfy the coverage criterion, and exact theory complexities will therefore depend on the examples given.

Examples for this task were generated using the correct theory for *illegal* given by Bain [2]. Figure 4 plots the performance of the $\mathcal{MC}$ measure on this task. The number of samples n in (6) was set to 2000. Examples were incrementally accumulated in a set, and the size of this set is represented by the horizontal dimension. The vertical dimension represents the compression ratio

$$1 - \frac{L(T|E)}{L(\perp|E)}$$

produced by a theory. Each theory is presented with the same example set. The graph clearly shows T_7 emerging as the preferred theory at 99 examples. Theory T_2 never compresses the data; it is too specific and its complexity after augmentation will be too high. Theory T_3 is a considerable improvement over T_2, being more general.

Figure 5 plots the same example data for the $\mathcal{PC}$ measure. The results show remarkable similarity, with the same preferred theory, T_7, emerging at the same point. Table 4 reproduces the data of Figures 4 and 5 at the point of 150 examples. Theory complexity $L(T)$ is measured after augmentation; the number of examples augmenting the theory are given in the second column. The ranking of theories given by each measure at 150 examples is as follows:

$\mathcal{MC}$	T_7	T_5	T_4	T_6	T_1	T_3	$\perp$	T_2
$\mathcal{PC}$	T_7	T_1	T_5	T_4	T_6	T_3	T_2	$\perp$

After 150 examples both methods have the same first choice T_7, the same last three choices (T_2, T_3 and $\perp$), and a closely-spaced group in the middle (T_1, T_4, T_5, T_6) that differs only in the placement of the general theory T_1.

3.3. Discussion

Based on these two examples, we can draw some tentative conclusions about the $\mathcal{MC}$ and $\mathcal{PC}$ complexity-based induction methods. It appears that the differences between the two measures is small so long as the theory only involves very shallow proofs: this is why they produce very similar results on the chess example. However, once proofs become nested, $\mathcal{PC}$ tends to break down in that it begins to favour overly–general theories. The cost of encoding proofs weighs against theories that require multi–step proofs. Indeed, even in the chess example, in which all proofs are shallow, theory $\top$ comes second in the $\mathcal{PC}$ ranking of theories, whereas for $\mathcal{PC}$ it comes fifth.

Nested proofs are typically caused by recursion. In such cases there will be correlations between parts of the proof, and between different proofs: failing to take these into account renders the $\mathcal{PC}$ coding overly redundant. For example, in all cases examined $L_{\mathcal{PC}}(E|T)$ is substantially greater than $L_{\mathcal{MC}}(E|T)$. The effect of

Table 3. Theories for the chess endgame task.

i	T_i
1	`illegal(_,_,_,_,_,_).`
2	`illegal(_,_,C,_,C,_).`

3	`illegal(_,_,C,_,C,_).`
	`illegal(_,_,_,D,_,D).`

4	`illegal(_,_,C,_,C,_).`
	`illegal(_,_,_,D,_,D).`
	`illegal(A,B,_,_,E,F) :- adj(B,F).`

5	`illegal(_,_,C,_,C,_).`
	`illegal(_,_,_,D,_,D).`
	`illegal(A,B,_,_,E,F) :- adj(B,F), adj(A,E).`

6	`illegal(_,_,C,_,C,_).`
	`illegal(_,_,_,D,_,D).`
	`illegal(A,B,_,_,E,F) :- adj(B,F), adj(A,E).`
	`illegal(A,B,_,_,A,B).`

7	`illegal(_,_,C,_,C,_).`
	`illegal(_,_,_,D,_,D).`
	`illegal(A,B,_,_,E,F) :- adj(B,F), adj(A,E).`
	`illegal(A,B,_,_,A,B).`
	`illegal(A,B,A,B,_,_).`

Table 4. Encoding 150 random examples of *illegal* using model complexity and proof complexity.

T	augs	$L(T)$	$L_{\mathcal{MC}}(E\|T)$	$L_{\mathcal{MC}}(T\|E)$	$L_{\mathcal{PC}}(E\|T)$	$L_{\mathcal{PC}}(T\|E)$
$T_1 = \top$	0	30	1827	1857	2700	2730
T_2	83	2121	1380	3501	1964	4084
T_3	23	634	1523	2156	2602	3235
T_4	4	195	1618	1814	2573	2768
T_5	5	232	1548	1780	2499	2731
T_6	5	258	1563	1821	2524	2782
T_7	0	154	1568	1722	2437	2592
$\perp$	0	3151	0	3151	1084	4235

this is that the complexity $L(T)$ of the theory itself is given less weight in the $\mathcal{PC}$ measure. A solution to this is to devise a coding method that takes account of proof correlations.

Both measures exhibit the desired behaviour on the chess endgame task, and $\mathcal{MC}$ also exhibits it on the network task. They begin by preferring the theory $\top$, but only until a certain number of examples is seen. This reflects the fact that there is a point up to which it is more economical to transmit the simple theory $\top$, sending all examples explicitly, than it is to transmit a more complex theory. This point was around 70 for the chess endgame task. By devising more efficient coding schemes for theories, the number could be reduced. Due to the apparent problems with $\mathcal{PC}$ on the network task, the causes of which are discussed above, it appears that $\mathcal{MC}$ may provide a better practical preference criterion for inductive logic programs.

4. Generalized model complexity

This paper has applied model complexity to the fundamental inductive inference task of (relational) concept learning. This section will suggest that the construction is just an instance of a generalized complexity–based technique.

Generalized model complexity is based on the following scenario. One has a logic program T, and wishes to efficiently communicate the model $M(T)$, that is, all ground facts entailed by the theory. There are two ways to do this. One is to simply transmit the logic program as a syntactical object, perhaps using some compression scheme such as the one outlined in Section 2.4. The other way is to generate a theory T' that is more general than T. The theory T' will tend to be less complex than T, and will therefore require fewer bits to encode. However, to enable the reconstruction of $M(T)$, it is necessary to specify the relevant subset $M(T') - M(T)$ of the model $M(T')$. This will require (according to $\mathcal{MC}$)

$$L_{\mathcal{MC}}(M(T)|T') \stackrel{\mathrm{def}}{=} \log_2 \binom{|M(T')|}{|M(T)|}$$

bits.[6] A statistical estimation procedure (Section 3.2) might be used to efficiently evaluate this expression. The description length — the total number of bits needed to communicate the model $M(T)$ — is

$$L(T') + L_{\mathcal{MC}}(M(T)|T').$$

The principle behind generalized model complexity is that theories are syntactical objects, and can be viewed as compact codings for Herbrand models.

It can be demonstrated that the model complexity scheme outlined in this paper is an instance of this generalized measure, and that an identical theory ranking would be produced by the generalized complexity principle. The impact of this result, and the tentative success of the earlier empirical analysis, suggests that inductive logic programming can be viewed as the *compression of models*. Negative examples will restrict the models under consideration, but are not essential to this formulation. Concept learning is only one very restricted type of inductive logic programming.

5. Conclusion

Complexity–based induction is an objective approach to evaluating induced theories, and is based on a very natural and intuitive principle. It requires that theory complexity and fit to examples be balanced. It reduces information storage requirements, but not to the extent that overgeneralization occurs. It asks for theories of just the right size.

This paper has demonstrated that complexity–based induction, which has for the most part been applied to propositional probabilistic theories and concepts, can be given two natural interpretations with respect to first–order logical induction. In particular, we have reviewed a proof complexity measure, and introduced a new model complexity measure for theory preference. These measures were applied to two problems of relational learning from positive–only examples.

In this paper we have considered the task of learning logic programs, and have not digressed into issues of noisy data and theory approximation. A treatment of these topics in the context of inductive logic programming can be found elsewhere [17, 20]. However, there are certainly real–world relational concepts which cannot be adequately modelled by deterministic logical theories, which can fail dramatically when the data include noise [8, 6]. To model these concepts will require some form of probabilistic predicate logic. Complexity–based induction can also apply with no modification to its basic prescription: the extraction of maximum redundancy from examples.

Both logical and probabilistic induction can benefit from research in complexity–based induction. Logic programs have a clear model theory, and can represent a much richer space of concepts than propositional logic. Inductive logic programming is clean and precise, relying on a well–defined model of generalization. On the other hand, research in machine learning of logical concepts can benefit from the large body of research on the minimum description length principle and Bayesian induction.

Acknowledgements

We thank the anonymous referees for their exceptionally thoughtful comments on an earlier draft of this paper.

This research has been supported by a Postgraduate Scholarship and an operating grant from the Natural Science and Engineering Research Council of Canada.

References

[1] D. Angluin. Inductive inference of formal languages from positive data. *Information and Control*, 45:117–135, 1978.

[2] M. Bain. Experiments in non–monotonic first–order induction. In S. Muggleton, editor, *Inductive Logic Programming*, pages 423–435. Academic Press, 1992.

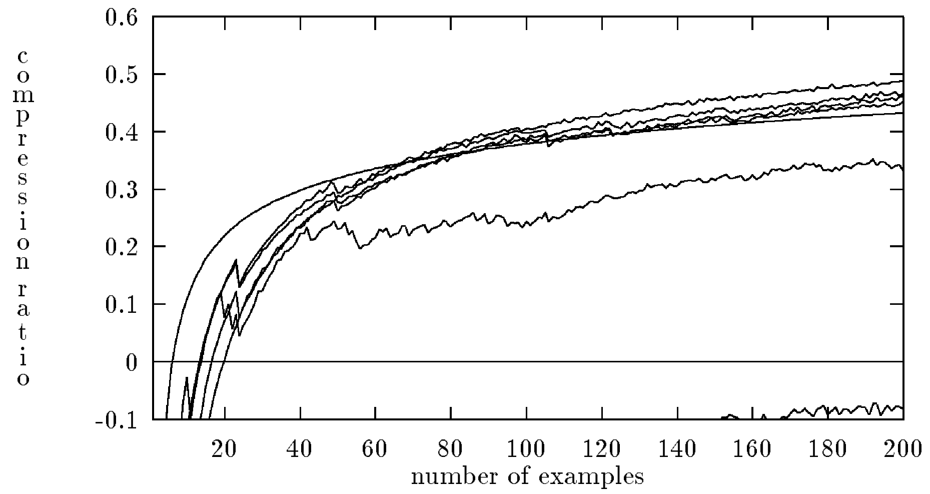

Figure 4. Evaluation of theories for chess: measure $\mathcal{MC}$.

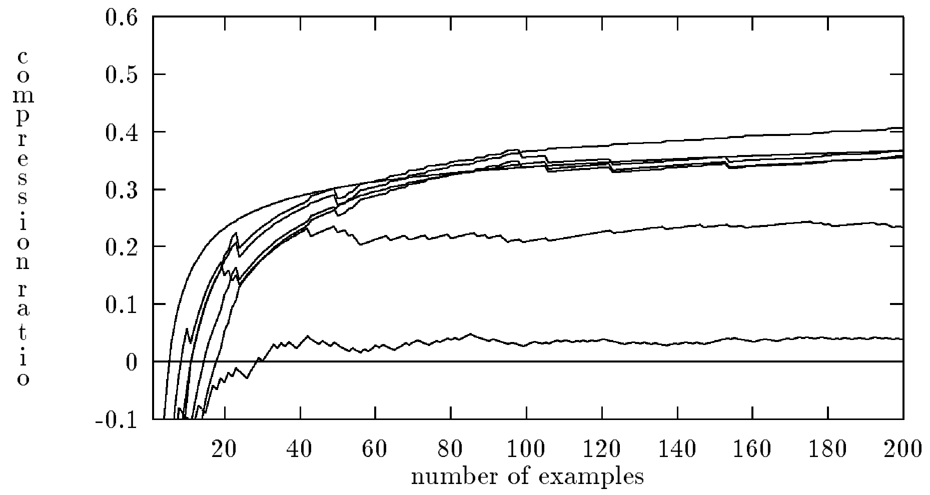

Figure 5. Evaluation of theories for chess: measure $\mathcal{PC}$.

[3] R. C. Berwick. Learning from positive–only examples. In R. Michalski, J. Carbonell, and T. Mitchell, editors, *Machine Learning: An Artificial Intelligence Approach*, volume II, pages 625–645. Morgan Kaufmann, 1986.

[4] W. Buntine. Generalized subsumption and its application to induction and redundancy. *Artificial Intelligence*, 36:149–176, 1988.

[5] G. J. Chaitin. *Information, Randomness & Incompleteness*. World Scientific, Singapore, 1987.

[6] P. Cheeseman. On finding the most probable model. In J. Shrager and P. Langley, editors, *Computational models of scientific discovery and theory formation*, chapter 3. Morgan Kaufmann, 1990.

[7] J. Feldman. Some decidability results on grammatical inference and complexity. *Information and Control*, 20:244–262, 1972.

[8] B. R. Gaines. Behavior/structure transformations under uncertainty. *Int. J. Man–Machine Studies*, 8:337–365, 1976.

[9] E. M. Gold. Language identification in the limit. *Information and Control*, 10:447–474, 1967.

[10] M. Li and P. Vitanyi. Inductive reasoning and Kolmogorov complexity. *J. Computer and System Sciences*, 44(2):343–384, 1992.

[11] J. W. Lloyd. *Foundations of logic programming*. Springer–Verlag, 1987.

[12] S. Minton. Quantitative results concerning the utility of explanation–based learning. *Artificial Intelligence*, 42:363–392, 1990.

[13] T. M. Mitchell. Generalization as search. *Artificial Intelligence*, 18:203–226, 1982.

[14] S. Muggleton. A strategy for constructing new predicates in first order logic. In *Proc EWSL 88*, pages 123–130, 1988.

[15] S. Muggleton. Inductive logic programming. In S. Muggleton, editor, *Inductive Logic Programming*, pages 3–27. Academic Press, 1992.

[16] S. Muggleton, editor. *Inductive Logic Programming*. Academic Press, 1992.

[17] S. Muggleton, A. Srinivasan, and M. Bain. Compression, significance and accuracy. In D. Sleeman and P. Edwards, editors, *Machine Learning: Proceedings of the Ninth International Conference (ML92)*, pages 338–347. Morgan Kaufmann, 1992.

[18] K. R. Popper. *The Logic of Scientific Discovery*. Hutchinson & Co. Ltd., 1959.

[19] J. R. Quinlan. Induction of decision trees. *Machine Learning*, 1:81–106, 1986.

[20] J. R. Quinlan. Learning logical definitions from relations. *Machine Learning*, 5(3):239–266, August 1990.

[21] J. Rissanen. Minimum description length principle. In S. Kotz and N. L. Johnson, editors, *Encyclopedia of Statistical Sciences*, pages 523–527. Wiley, 1985.

[22] C. Sammut and R. B. Banerji. Learning concepts by asking questions. In R. Michalski, J. Carbonell, and T. Mitchell, editors, *Machine Learning: An Artificial Intelligence Approach*, volume II, pages 167–191. Morgan Kaufmann, 1986.

[23] E. Y. Shapiro. *Algorithmic program debugging*. The MIT Press, 1983.

[24] R. J. Solomonoff. Complexity–based induction systems: Comparisons and convergence theorems. *IEEE Trans. Information Theory*, IT–24(4):422–432, 1978.

[25] L. Sterling and E. Shapiro. *The Art of Prolog*. The MIT Press, 1986.

[26] C. S. Wetherell. Probabilistic languages: A review and some open questions. *ACM Computing Surveys*, 12(4):361–379, December 1980.

[27] P. H. Winston. Learning structural descriptions from examples. In P. H. Winston, editor, *The Psychology of Computer Vision*. McGraw-Hill, 1975.

[28] I. H. Witten, R. Neal, and J. G. Cleary. Arithmetic coding for data compression. *Communications of the ACM*, 30(6):520–540, June 1987.

Notes

1. In this paper we do not consider the "constructive" induction scenario when inductive inference can modify the Herbrand base of a theory.
2. Throughout this paper, we assume an efficient technique such as *arithmetic coding* [28], which can code an integer from 1 to k in approximately $\log_2 k$ bits, and an event with probability p drawn from a discrete distribution in approximately $-\log_2 p$ bits. It must be emphasized that the physical communication of the theory and examples is just a metaphor, and we are only interested in the evaluation of theories. That is, we are not overly concerned with the production of the string $D(T) \cdot D(E|T)$.
3. This is a slight variant of the proof complexity measure given in [17]. In particular, if negative examples are to be communicated, $|E|$ bits would be added to the expression: 1 bit for each example indicating its polarity. However, we are interested in comparing this proof complexity measure with our model complexity measure in situations where positive-only examples are available. Also, a constant quantity of $\log_2 |E|$ could be added to every theory to identify the number of examples.
4. Note that this estimation technique cannot be used if the actual transmission of the examples is desired. However, in the context of theory evaluation this is not an issue.
5. Note the anonymous variables in the theories. Each occurrence of an anonymous variable is distinct, thus the singleton rules of theories T_1 and T_2 have 6 and 5 non–generative variables, respectively.
6. Note that the evaluation of this expression requires that both $M(T')$ and $M(T)$ are finite. This necessitates the restriction to function–free logic programs.