Learnability of Kolmogorov-Easy Circuit Expressions Via Queries

José L. Balcázar[1], Harry Buhrman[2], and Montserrat Hermo[3]

[1] Dept. LSI, Edif. FIB, Universitat Politécnica de Catalunya, 08071 Barcelona, Spain
[2] CWI, Kruislaan 413, Amsterdam, The Netherlands
[3] Facultad de Informatica de San Sebastián, Universidad del País Vasco, Spain

Abstract. Circuit expressions were introduced to provide a natural link between Computational Learning and certain aspects of Structural Complexity. Upper and lower bounds on the learnability of circuit expressions are known. We study here the case in which the circuit expressions are of low (time-bounded) Kolmogorov complexity. We show that these are polynomial-time learnable from membership queries in the presence of an NP oracle. We also exactly characterize the sets that have such circuit expressions, and precisely identify the subclass whose circuit expressions can be learned from membership queries alone. The extension of the results to various Kolmogorov complexity bounds is discussed.*

1. Introduction

This paper presents algorithms to learn circuit expressions in the "learning via queries" model of Computational Learning. In this model, the learning algorithms interact with their environment, trying to grasp a concept. The concept will be formally modeled as a set of encodings, which are themselves simply binary strings. The interaction is formalized by "queries", and these in turn are of the kind of questions suggested by the set-theoretic modeling of concepts. For instance, the simplest one is the "membership" query, in which the learner presents a binary string and asks for its classification as "in" or "out" the concept set; and the "subset" query presents (a finite encoding of) a set of words and asks whether that set is a subset of the concept set. We focus on the "bounded learning" model from [11], where it is not necessary to completely identify the concept: a length bound is given initially to the learner, and it must identify the concept up to that bound.

Concepts are usually represented in some manner. We study here the representation via circuit expressions. These are defined inductively just as regular expressions, by the operations of union, concatenation, and Kleene star, with

* Work partially supported by the EC through the Esprit program (project 7141, ALCOM-II, and Working Group 8556, NeuroColt), and through the HCM program (project CHRX-CT93-0451, COLORET Network), by the Ducth NWO, and by the Spanish DGICYT (project PB92-07099). E-mail: `balqui@lsi.upc.es`, `buhrman@cwi.nl`, `jiphehum@si.ehu.es`

the difference that boolean circuits are considered also as circuit expressions. We can therefore take unions of circuits, or Kleene stars of circuits. Their meaning is defined in the standard way (but see below for a complete description).

Circuit expressions were introduced in [12] to provide a natural link between Computational Learning and certain aspects of Structural Complexity. They share most interesting properties of the boolean circuit model, but overcome the inconvenience that a fixed boolean circuit has a fixed number of inputs and, therefore, accepts a subset of $\{0,1\}^n$ for some fixed n; circuit expressions may even accept infinite sets by using the Kleene star and union. In [12], learnability issues for this and other representation classes were related to the "representation finding" problem of Structural Complexity, and this allowed for a precise definition of the "computational power" of a learning protocol, given in terms of (relativizations of) the polynomial-time hierarchy.

For instance, following the intuitions there, it can be seen that it is possible to learn circuit expressions for a concept A via membership queries if and only if A is polynomial-time T-equivalent to a tally set. (This is argued in some more detail below.) The polynomial-time tally T-degrees, which were known to correspond to the so-called "self-producible circuits", characterize therefore the concepts for which a specific learnability problem is solvable. This paper will pursue further this sort of connections.

Upper and lower bounds on the learnability of boolean circuits are known, and most of them carry over to circuit expressions. Actually, in the work made up to now, the distinction between circuits and circuit expressions is hardly worth to be made, being at most a small technical detail. Research in Structural Complexity shows that circuits for a set can be obtained from an oracle in Σ_2^p plus an NP oracle relative to the set itself [8]. It is also known that it is possible to learn deterministically boolean circuits from equivalence queries plus a Σ_3^p oracle, or just with an NP oracle by a randomized algorithm [5]. Although there is motivation for the use of equivalence queries (e.g. connections to the "mistake-bounded" model), these fall into the category of "expensive" query types, in that for many applications it makes little sense to expect a teacher to answer equivalence queries.

However, for reasonably simple queries like membership, it is not difficult to see that circuits cannot be identified with polynomially many queries, regardless of the amount of computational power available [1]. As for other queries, in [12] the computing power needed to learn a "repetitive" variant of circuits is exactly characterized by classes in the relativized polynomial time hierarchy. The survey [6] gives a precise account of many such results and related ones.

We study here the case in which the circuit expressions are of low time-bounded Kolmogorov complexity; specifically, the case in which they have logarithmically long descriptions, from which the (polynomial size) expressions can be recovered in polynomial time. There are two reasons. First, in many natural cases, this would be satisfactory enough, since frequently large circuits are built of replications of small ones; for instance, the quadratic circuit simulation of a time-bounded Turing machine is a very regular circuit consisting of a repeated fized-size pattern, and its Kolmogorov complexity is precisely logarithmic. Second, it is known from other approaches to Computational Learning (such as PAC) that sometimes concepts that are not (or not known to be) learnable in general become learnable under such "simplicity" conditions [9]. This paper proves that this is also the case with learning circuit expressions via queries.

First, we exactly characterize the sets that have such logarithmically easy circuit expressions, in order to know exactly what concepts are we fighting with. The proof is not immediate, but the difficulties can be solved by using techniques developed previously by the authors.

To analyze the learnability of these concepts, we point out some easy observations proving that they are PAC-learnable, and then we study the learnability from queries. We start from a naive prefix-search algorithm that uses, in a standard way, a "relativized NP" oracle, similar to the ones used in [12] for the general case. Then we show how to use membership queries to "un-relativize" the NP oracle. This means that we prove that for every set having logarithmically easy circuit expressions, these can be found with membership queries in the presence of an NP oracle, in polynomial time.

An interesting observation is that the analysis and the replacements of the oracles bear also consequences for the Structural Complexity of the classes studied, yielding so-called "lowness" properties for all such sets. Also, it is worth pointing out that our algorithms work by directly obtaining a logarithmically long seed that will produce, in polynomial (actually linear) time, a circuit expression for the concept.

A natural subclass to try to understand is that of those concepts for which the learnability can be performed using simply the most inexpensive queries, those of "membership", without resorting to additional oracles. We study this subclass and characterize it in terms of polynomial time degrees: for a concept A, circuit expressions can be learned from membership queries alone if and only if A belongs to a polynomial-time doubly tally T-degree (see preliminaries for a definition). This class was already known, and characterizations of it, with an analysis of its inner structure, appear in [7].

The extension of the results to various Kolmogorov complexity bounds is also discussed. We prove that many of the technical properties on which our results are based also hold for other bounds, in particular for the polylog case. We extend some of the characterizations to these bounds. An interesting fact is that the corresponding conditions on the tally sets are no longer qualitative but quantitative: from $\log^2 n$ onwards, the tally sets must be defined by density conditions, while for the $\log n$ case we had to request a specific pattern of the words in the corresponding tally sets. The deep reason of this divergence is not fully understood.

2. Preliminaries

Complexity Theory

Our notions and notations of Complexity Theory are standard; see [2]. Our sets consist of words over a single fixed alphabet with at least two symbols; we frequently assume that the alphabet is $\{0,1\}$, to operate on words with boolean models of computation. The most basic of them are assumed known. For a set A, $A^{=n}$ is the set of words of length n in A, and $A^{\leq n}$ is the set of words of length up to n in A.

Sets of sets are usually called classes. The class P consists of the problems solvable deterministically in polynomial time. If the polynomial time computation has access to an oracle set A, the resulting class is denoted P^A. When $B \in \mathrm{P}^A$, we say that B is polynomial time Turing reducible to A; if both are each reducible to the other, then we say that they are polynomial time Turing *equivalent*. For C a complexity class C^A denotes the class where each machine in the class has acces to the oracle A. The class of all sets polynomial time Turing equivalent to A is denoted $E_T(A)$; this class is the (polynomial time Turing) *degree* of A. We use similarly a whole class of sets instead of a single set A: so, $E_T(C)$ is the class of sets polynomial time Turing equivalent to some set in the class C.

We denote by SAT any NP-complete problem such as the well-known "Satisfiability" problem of boolean formulas. A set S is sparse if the cardinality of $S^{\leq n}$ is bounded by some polynomial. A set T is tally if $T \subseteq \{0\}^*$. A set is doubly tally if it is tally and all of its words have length a power of 2. The class of all doubly tally sets is denoted by Tally2.

Oracles are one way of providing computational devices with additional information. Advice words are another, and give rise to nonuniform complexity classes. The nonuniform class Full-P/log consists of all sets for which there exist advice words w_n with $|w_n| \leq c \log(n)$ for some constant c, and a set $B \in \mathrm{P}$, such that for all n it holds:

$$\forall x(|x| \leq n),\ (x \in A \iff \langle x, w_n \rangle \in B)$$

Other advice classes are obtained using other families of bounds for the length of the advice words. For instance, using polynomials, we get P/poly, which is known to coincide with the problems that can be solved by having polynomial size circuits. An equivalent definition is $\bigcup_{S \text{ is sparse}} \mathrm{P}(S)$. A similar, less known characterization of Full-P/log is as follows:

Theorem 1. *[4]* Full-P/log $=$ P(Tally2).

Circuit expressions were introduced in [12]. They are constructed exactly as regular expressions, with the additional proviso that boolean circuits are circuit expressions. Hence, a circuit expression is a boolean circuit with a single output, or the sum of two circuit expressions, or a star operator applied to a circuit expression. The language denoted by a circuit expression is defined in the usual way: if the expression is a standard circuit, it denotes the set of words for which the output is 1; the sum corresponds to the union, and the star to the stardard Kleene star operator of regular languages. Circuit expressions can be evaluated in polynomial time [12].

We denote by $CEX(A, n)$ the set of circuit expressions C such that $L(C)$ coincides exactly with A up to size n.

Bounded Learning

Following [11], we model concepts as sets of words. We follow the "bounded learning" framework from that reference. The learner is a (usually polynomial-time) algorithm, sometimes with access to some oracle set. It is initially provided with a bound n on the size of the representation of the target concept, and with a length bound m; its goal is to output a representation, of size no larger than n, that correctly represents the target concept up to size m. Information about the target concept is obtained from a "teacher" able to somehow compute the answer to some selected set-theoretic questions. Basic queries are Membership ("Is this word a member of the target set?") and Equivalence ("Does this representation correctly represent the target set up to the length bound?"); less frequent queries are Subset and Superset. Sometimes counterexamples are required for some of these queries, but in this paper we will not use this brand. See [11] for the exact notion of representation class and for precise formalizations of all these concepts.

We say that a class of sets is (Mem)-CEX-learnable if there is a (single) learning algorithm that learns circuit expressions for all sets in the class, using membership queries, in polynomial time. Sometimes an oracle (e.g. SAT) is assumed to be available, and we will explicitly indicate this fact.

Obviously a polynomial time learner only can write down as output a circuit expression of polynomial size, so that each CEX-learnable concept is in P/poly. The following is also immediate:

Proposition 2. *If $\mathcal{C}_1 \subseteq \mathcal{C}_2$ and $\mathcal{C}_2$ is (Mem)-CEX-learnable, then $\mathcal{C}_1$ is (Mem)-CEX-learnable.*

The connection with the theory of polynomial time degrees comes from the following fact, pointed out by Osamu Watanabe (personal communication). Results in the same spirit (for more general representation classes) appear in [12].

Proposition 3. *If $\mathcal{C}$ is (Mem)-CEX-learnable then every concept in $\mathcal{C}$ is polynomial-time T-equivalent to some tally set.*

The proof follows the guidelines of the characterization of the polynomial-time tally T-degrees by means of "self-producible circuits"; see for instance [2]. We simply sketch it here. Essentially, given a learnable set A, a tally set can encode in a standard way the circuit expression that the learning algorithm finds. Given the set as oracle, the tally set can be decided by running the learner and checking that the encoding is correct; conversely, given the tally set as oracle, the corresponding circuit expression can be retrieved and evaluated.

Conversely, if A is T-equivalent to a tally set T in polynomial time, a learning algorithm on input n scans all of 0^* up to some polynomial $p(n)$ and, for each word, uses queries to A to decide membership to T. Once a large enough initial segment of T is known, it is easy to design a polynomial size circuit expression that simulates the machine reducing A to T on inputs of length up to n, which has built-in enough information about T to compute A without further aid.

3. Learning easy circuit expressions

In order to give a precise definition of the subclass of circuit expressions we study, we introduce the resource-bounded Kolmogorov complexity classes. (See [10] for undefined notions and properties.) Fix any universal Turing machine U. Define the sets of bounded Kolmogorov complexity strings $K[f(n), g(n)]$ as follows:

Definition 4. $x \in \mathrm{K}[f(n), g(n)]$ if there exists y, $|y| \leq f(|x|)$, such that $U(y) = x$ in at most $g(|x|)$ steps.

We focus here in the class K[log,poly] in which functions from $O(\log n)$ are selected for f, and functions from $n^{O(1)}$ are selected for g. The chosen constants may depend on the set to be learned, but not, of course, on the length for which the circuit expression is desired.

Definition 5. A set A has easy circuit expressions if and only if for some constant c, and for each n, there is $C_n \in CEX(A, n)$, of size $|C_n| \leq n^c$, such that $C_n \in \mathrm{K}[c \log n, n^c]$.

Before moving into learning such easy circuit expressions, it is worth to know something about the sets having easy circuit expressions. We have:

Theorem 6. *A set A can be decided by a family of easy circuits expressions if and only if $A \in$ Full-P/ log.*

Proof (Sketch). Suppose first that $A \in$ Full-P/ log: there exist a sequence of advice words w_n with $|w_n| \leq c\log(n)$ for some constant c, and a set $B \in$ P, such that for all n it holds:

$$\forall x(|x| \leq n),\ (x \in A \iff \langle x, w_n \rangle \in B)$$

Using standard techniques, it is easy to construct a polynomial size circuit $C_n \in$ K[log, poly] that recognizes $A^{=n}$. These can be put together into an easy circuit expression for $A^{\leq n}$. It is important to notice here that the logarithmically many constant gates encoding the advice word are *the same* for all the circuits in the circuit expression.

Suppose now that A has easy circuit expressions; thus there exists $p(n)$ such that for all n there exists a circuit expression CE_n in such a way that $CE_n \in CEX(A,n)$, $|CE_n| \leq p(n)$, and $CE_n \in \mathrm{K}[\log, \mathrm{poly}]$. Consider the set $B \in \mathrm{P}$ formed by pairs $\langle x, s\rangle$ where x is in the language recognized by the circuit expression produced by U from s. The running time of U is only allowed to be an appropriate polynomial.

Here is where the proof becomes nontrivial. It is easy to see that it is not enough to take simply the seeds as advice words. If, together with x, we give a seed of roughly the same length, it takes exponential time to find and evaluate an exceedingly large circuit expression; and, if we forbid this possibility, we are not fulfilling the definition of Full-P/ log. The advice words to be taken are, instead, concatenations of the seeds for circuit expressions C_n for n a double power of 2. This technique was introduced in [4], and a similar one is used below for another characterization theorem (see the proof in the appendix). In the pairs $\langle x, s\rangle \in B$, now s is a concatenation of self-delimiting descriptions for several circuit expressions, and at least one of them is polynomial on $|x|$ and can be used to decide membership into A. Further details will be provided in the full text. $\square$

This last proof being completely constructive, it also indicates that in order to learn circuit expressions it is enough to find a way of computing the appropriate advice words. These, together with the length up to which the circuit expression is desired and a constant size program, are enough to fully reconstruct the desired circuit expression.

We move now to discuss learnability. Fix $A \in$ Full-P/ log. First we allow ourselves the use of a set in NP^A, in two different forms. From the fact that only polynomially many descriptions exist for easy circuit expressions, we obtain immediately an algorithm based on equivalence queries: simply, reconstruct all potential circuit expressions by cycling over all logarithmically long seeds, and ask each one as an equivalence query. It is easy to see that the equivalence query (which does not need here a counterexample) can be answered by an oracle in NP^A. This obvious algorithm proves not only that equivalence queries suffice, but also, via a now standard transformation [1], that easy circuit expressions are PAC-learnable.

A serious objection is that allowing a number of equivalence queries similar to the number of potential hypothesis does not seem reasonable. But it is not difficult to reduce the number of queries to a different set in NP^A.

Theorem 7. *Easy circuit expressions for every set $A \in$ Full-P/ log can be learned in polynomial time, making logarithmically many queries to NP^A.*

Proof. Suppose $A \in$ Full-P/ log. We will show how to construct the advice words, and then appeal to the constructive character of the previous characterization.

We will say that the word y, with $|y| \leq c \log n$, is a "good advice" if $\forall u\ (|u| \leq n\ (\langle u, y\rangle \in B \iff u \in A)$. This predicate is in co-NP^A. Let GA be the following oracle set:

$$GA = \{\langle z, 0^n\rangle \mid |z| \le c \log n \wedge \exists y\, z \sqsubseteq y\, (|y| = c \log n \wedge y \text{ is a "good advice"})\}$$

The set GA belongs to co-NP^A, and allows for a prefix-search procedure to calculate a good advice by asking GA about consecutive prefixes. Thus, if $A \in$ Full-P/log then circuit expressions for A can be learned in polynomial time, making logarithmically many queries to NP^A. □

Observe that trivially the converse also holds.

Our interest centers now, therefore, on learning via membership queries to the target concept. Actually, the algorithm used is somewhat more general, in that it allows us to prove that, for $A \in$ Full-P/log, any set in NP relativized to A can be decided in polynomial time with queries to A directly, in the presence of a set in (unrelativized) NP. Specifically, we are going to prove that, for all sets $A \in$ Full-P/log, it holds that $\mathrm{NP}^A \subseteq P^{A \oplus SAT}$.

Theorem 8. *If $A \in$ Full-P/log, then $\mathrm{NP}^A \subseteq P^{A \oplus SAT}$.*

Proof (Sketch). Let A be a set in Full-P/log, and let M be a nondeterministic polynomial time oracle Turing machine to be simulated on some input x of length n. The kernel of the algorithm is a function that, given two potential advice words t_i, t_j, tries to find a word on which they give different answers; this is where the NP oracle is used. If two advice words t_i and t_j give always the same answers, either both are correct of both are wrong, so it is unnecessary to keep both; one is discarded. For the remaining pairs, simply make membership queries about the distinguishing word; the advice that is inconsistent with the answer gets discarded. Only the correct one, that exists by hypothesis (and is unique after discarding advices that give the same answers) will emerge out of the game.

The function that uses an NP oracle to find a word that distinguishes two advice words follows a standard prefix search procedure. □

In the terminology of structural complexity, this theorem is a result on "lowness"; more precisely, the sets A for which $\mathrm{NP}^A \subseteq P^{A \oplus SAT}$ form the first level of the extended low hierarchy, and are denoted as EL_1^{Σ}. Thus this theorem is actually showing that Full-P/log $\subseteq EL_1^{\Sigma}$. We immediately obtain:

Corollary 9. *Easy circuit expressions for every set $A \in$ Full-P/log can be learned in polynomial time with membership queries and an NP oracle.*

There is one more observation to point out: we have now two ways of obtaining a learning algorithm using subset and superset queries. On the one hand, these can simulate equivalence queries in the obvious manner. On the other hand, as shown in [5], when learning a representation class based of boolean models, subset and superset are able to simulate an NP oracle, and of course a membership query. This yields two different learning algorithms using superset and subset queries.

4. Learnability from membership queries only

The purpose of this section is to precisely characterize those concepts for which circuit expressions can be learned in polynomial time using only membership queries, without the additional NP oracle. Note that to construct in polynomial time easy circuit expressions is equivalent to finding their logarithmically long seeds, since these can be found, if necessary, in polynomial time by exhaustive search.

Theorem 10. *The following facts are equivalent:*

i/ $A \in E_T(\text{Tally2})$.

ii/ A is decided by a family of easy circuit expressions whose descriptions can be obtained in polynomial time using membership queries.

Proof (Sketch). First we prove i/ $\Longrightarrow$ ii/. Let A be a set in $E_T(\text{Tally2})$, and let $T \in \text{Tally2}$ be such that $T \in \mathrm{P}^A$ and $A \in \mathrm{P}^T$. Therefore $A^{\leq n}$ can be decided from the characterstic sequence of T up to n^c for some c. As T is very regular, its characteristic sequence can be expressed relative to the set $\{0^{2^n} \mid n \geq 0\}$ and then the size of this information is logarithmic in n. From this, and the fact that $A \in \mathrm{P}^T$, it follows that A has easy circuit expressions as in the characterization given in the previous section. Moreover, these can be constructed from the list of words of T up to length n^c, and this can be obtained from membership queries to A.

In the second place we see that ii/ $\Longrightarrow$ i/. Let A be a set recognized by a family of easy circuit expressions, and assume that the logarithmically long seeds of the expressions can be obtained in P^A. Invoking again the techniques of [4], it is possible to prove that there exists a (different) family of seeds for circuit expressions for A with the property that each seed is a prefix of the next one. These can be easily constructed from the old ones, so that we can keep the information of the seeds in a tally2 set T that is in P^A. On the other hand, $A \in \mathrm{P}^T$ because from the information in T we can construct in polynomial time a circuit expression to decide A. □

5. Polylogarithmic complexity

Of course, it would be interesting to relax the condition of "easy" circuit expressions on which all the constructions are based. Unfortunately, we cannot extend all the results for bounds larger than logarithmic. But, for certain reasonable conditions on the bounds, some parts of the characterization carry through; and, concentrating on polylogarithmic bounds, there are alternative characterizations that extend those given in the last section.

We first observe that a more careful tuning of the proof technique from [4] gives the following result. The previous section has used extensively the fact that it is possible to find a family of advices for each set in Full-P/ log with the property that all of them are prefixes of a single infinite word [4]. This class is called Pref-P$/O(f(n))$. This fact can be extended as follows:

Theorem 11. *If $f(n)$ is an unbounded function that can be calculated in polynomial time, then* Full-P$/O(f(n)) =$ Pref-P$/O(f(n))$.

The proof is given in the appendix.

We will focus next in the nonuniform classes defined from polylog advices, that is, Full-P$/O(\log^i(n))$. The next question we address is how to extend the characterization of the sets with easy circuit expressions learnable only from membership queries, to not-so-easy (e.g. polylog-easy) circuit expressions. Observe that nowhere in the class E_T(Tally2) appears explicitly a logarithm that could be changed into a polylog function. Actually, as it turns out, there is a nice reason for this.

Indeed, as we shall see shortly, there is a marked difference between the logarithmic and the polylogarithmic case. Both allow for a characterization in terms of polynomial-time Turing degrees of tally sets. In the case discussed in the previous section, the tally sets used were defined by a "qualitative" condition that the words were only of certain fixed lengths, namely powers of 2. When we move to polylog functions, the characterization no longer has this flavor, but it instead becomes purely "quantitative": it corresponds to tally degrees of polylog density. Moreover, the exponent of the density tightly corresponds to the exponent of the complexity of the circuit expressions, modulo a log factor; and this log factor precludes the use of this characterization for the logarithmic case. So the theorem of the previous section cannot be obtained as a particularization of the present one.

Let us denote as $\mathcal{F}$-Tally the class of tally sets of density bounded by a function from $\mathcal{F}$, i.e., having at most $f(n)$ words up to length n where $f \in \mathcal{F}$.

Lemma 12. *For all $i \geq 1$, the following holds:*

$$\text{Full-P}/O(\log^{i+1}(n)) = \mathrm{P}_T(O(\log^i(n))\text{-Tally}).$$

Proof (Sketch). We show first $\mathrm{P}_T(O(\log^i(n))\text{-Tally}) \subseteq \text{Full-P}/O(\log^{i+1}(n))$. In polynomial time, at most a polynomially long initial part of the tally set is accessible, and we need to know the tally oracle T up to that length. This information fits into $O(\log^{i+1}(|x|))$ bits, and constitutes our advice word.

Conversely, we see that $\text{Full-P}/O(\log^{i+1}(n)) \subseteq \mathrm{P}_T(O(log^i(n))\text{-Tally})$. By the previous theorem, it suffices to prove it for Pref-P$/O(\log^{i+1}(n))$. We encode the single infinite sequence limiting the successive advices into a tally set of the indicated density.

The encoding scheme is as follows. Let T be a tally set of $k\log^i(n)$ density, where k is a constant. In the interval between 0^{2^n} and $0^{2^{n+1}}$, there can be at most $k(n+1)^i - kn^i \leq rn^{i-1}$ possible new words belonging to T, for some constant r.

These must be chosen among the 2^n words of the interval. So, we divide the 2^n words into rn^{i-1} parts. Each part has the following number of words:

$$\mathcal{N} = \lfloor \frac{2^n}{rn^{i-1}} \rfloor$$

So, we can interpret each part as a digit in base $\mathcal{B} = \mathcal{N} + 1$, depending on where the only word of that part is, and encode a number with rn^{i-1} such digits. Straightforward computation shows that there is enough room to encode the needed part of the infinite sequence limiting the advices. The advice can be easily decoded in polynomial time. □

Theorem 13. *For all $i \geq 1$, the class of sets having polynomial size circuit expressions in* $\mathrm{K}[O(\log^{i+1}(n)), \mathrm{poly}]$ *whose descriptions are learnable in polynomial time from membership queries is precisely* $E_T(O(\log^i(n))\text{-Tally})$.

Proof (Sketch). Use the encoding described in the lemma to code into a tally set the seeds for the circuit expressions found by the learning algorithm. Not all of them are to be encoded: those are to be selected as in the proof given in the appendix. (Details are again deferred to a complete version of the paper.) This yields a tally set that is T-equivalent to the learned concept. The converse follows exactly as in the proof of the main result of the previous section. □

Taking the union over all i gives:

Corollary 14. *The class of sets that have a polynomial size circuit expression in* $\mathrm{K}[O(\log^{O(1)}(n)), \mathrm{poly}]$*, and whose description is learnable in polynomial time from membership queries, is precisely* $E_T(O(\log^{O(1)}(n))\text{-Tally})$.

6. Appendix

We present here the proof of theorem 12. The proof technique is the same, or simpler, for the omitted proofs of theorems 7, 11, and 14, and is a generalization of that used in [4] to prove theorem 1. A similar construction appears in [3], the essential difference being that in their context the time bounds for the Kolmogorov complexity definitions are based on the length of the output, and this requires a number of additional hypothesis and extra information manipulation.

The nontrivial inclusion is from left to right. Let $A \in \text{Full-P}/O(f(n))$, so that there exists a Turing machine M that decides A in polynomial time, with the help of advices $\{w_n\}_{n\in\mathbb{N}}$. There exists a function $h(n) \in O(f(n))$ such that each w_n has the length bounded by $h(n)$. Without loss of generality we assume that h is unbounded too. We have to construct an infinite sequence, in such a way that the prefixes of length $O(f)$ can be used as well as advices for A.

Using the fact that any advice corresponding to a particular length can be used by smaller lengths, the infinite sequence could be the concatenation of some w_n's. The idea is to keep only the information w_n for some selected n's instead of storing all of them. In order to choose the w_n's we have to consider a balance between two contradictory restrictions. If we select w_n's too frequently, then they will need too many bits and the piece of the infinite sequence could be too large; but if we select them too separated, then for some words the nearest valid advice would be too long to be extracted from the infinite sequence in polynomial time. It turns out that there is a way of skipping w_n's for which the balance is satisfactory.

Since h is unbounded, it is possible to find for each n a number m_n such that $2^{m_n-1} < h(n) \leq 2^{m_n}$; moreover, given n, the search of m_n is done in polynomial time. Remark that eventually many n's are associated to the same m_n. Let $g(n)$ be the following function:

$$g(n) = \max\{i \mid h(i) \leq 2^{m_n}\}$$

Construct an infinite sequence β including all of $\{w_{g(n)}\}_{n \in N}$. In order to get a good manipulation of the sequence, we construct β step by step, appending a power of two many bits each time. Namely $\beta = p_1 p_2 p_3 \dots p_n \dots$ in such a way that $\forall i > 0$ holds:

$$p_i = \begin{cases} \underbrace{w_{g(n)} 10 \dots 0}_{2^{m_n}} & \text{if } 2^i = 2^{m_n} \text{ for some } n \\ \underbrace{0 \dots 0}_{2^i} & \text{otherwise} \end{cases}$$

The minimum prefix of β containing the information of the advice $w_{g(n)}$ will be exactly $\beta_{1:2^{m_n+1}-1}$, which denotes the finite word made up with the bits of β starting at 1 and ending at $2^{m_n+1} - 1$. Then A is in Pref-P$/O(f(n))$ because the following algorithm decides whether a word x is in A.

input x;
$n := |x|$;
Find m_n;
From $\beta_{1:2^{m_n+1}-1}$ obtain $\beta_{2^{m_n}:2^{m_n+1}-1}$;
Discard the 10^* tail, obtaining advice word w
Simulate $M(\langle x, w \rangle)$ and accept if and only if M accepts.

The amount of bits from β used to decide whether x belongs to A is $\beta_{1:2^{m_n+1}-1}$. Its length is exactly $2^{m_n+1} - 1$. From the fact that $2^{m_n-1} < h(n)$, we obtain that $m_n < \log(h(n)) + 1$, so $2^{m_n+1} - 1$ is bounded by $4h(n)$. Therefore, $A \in$ Pref-P$/O(f(n))$. $\square$

References

1. D. Angluin: "Queries and Concept Learning". *Machine Learning* **2** 1988, 319–342.
2. J. L. Balcázar, J. Díaz, J. Gabarró: *Structural Complexity I.* Springer Verlag 1988.
3. J. L. Balcázar, R. Gavaldà, M. Hermo: "On infinite sequences almost as easy as π". *Workshop on Applications of Descriptional Complexity*, Rutgers Univ., july 1994. Report LSI–94–24–R, Univ. Politècnica de Catalunya.
4. J. L. Balcázar, M. Hermo, E. Mayordomo: "Characterizations of logarithmic advice complexity classes". In: *Algorithms, Software, Architecture: Information Processing 92*, Elsevier 1992, vol. 1, 315–321.
5. N. Bshouty, R. Cleve, S. Kannan, C. Tamon: "Oracles and queries that are sufficient for exact learning". In: *Conference on Computational Learning Theory* 1994, 130–139.
6. R. Gavaldà: "The complexity of learning with queries". In: *Structure in Commplexity Theory* 1994, 324–337.
7. R. Gavaldá, O. Watanabe: "Structural analysis of polynomial time query learnability". *Mathematical Systems Theory* **27** 1994, 231–256.
8. M. Hermo: "Degrees and reducibilities of easy tally sets". In: *Mathematical Foundations of Computer Science 94*, Springer-Verlag Lecture Notes in Computer Science 8411994, 403–412.
9. J. Köbler: "Locating P/poly optimally in the low hierarchy". In: *Symp. Theoretical Aspects of Computer Science 93*, Springer-Verlag Lecture Notes in Computer Science 665 1993, 28–37.
10. M. Li, P. Vitányi: "Learning simple concepts under simple distributions". *SIAM Journal on Computing* **20** 1991, 911–935.
11. M. Li, P. Vitányi: *An introduction to Kolmogorov complexity and its applications.* Springer-Verlag 1994.
12. O. Watanabe: "A framework for polynomial time query learnability". *Mathematical Systems Theory* **27** 1994, 211–229.

This article was processed by the author using the TEX macro package from Springer-Verlag.