’Machine Learning’ manuscript No.
(will be inserted by the editor)

The Teaching Size: Computable Teachers and Learners for Universal Languages

Jan Arne Telle · José Hernández-Orallo ·
Cèsar Ferri

Received: date / Accepted: date

Abstract The theoretical hardness of machine teaching has usually been analyzed for a range of concept languages under several variants of the teaching dimension: the minimum number of examples that a teacher needs to figure out so that the learner identifies the concept. However, for languages where concepts have structure (and hence size), such as Turing-complete languages, a low teaching dimension can be achieved at the cost of using very large examples, which are hard to process by the learner. In this paper we introduce the *teaching size*, a more intuitive way of assessing the theoretical feasibility of teaching concepts for structured languages. In the most general case of universal languages, we show that focusing on the total size of a witness set rather than its cardinality, we can teach all total functions that are computable within some fixed time bound. We complement the theoretical results with a range of experimental results on a simple Turing-complete language, showing how teaching dimension and teaching size differ in practice.

Keywords Machine Teaching, Teaching Dimension, Teaching Size, Compression, Universal Languages, P” programming language, Levin’s search.

1 Introduction

Learning from examples when the concept class is rich and infinite is hard. Given a concept, the teacher is faced with the problem of finding a set of examples, called the witness set, that allows the learner to uniquely identify the concept. The teaching dimension of a concept is the minimum cardinality of a witness set for the concept. The teaching dimension of the whole class is the maximum teaching dimension of any concept in the class. When the class is rich and infinite this maximum will usually be unbounded (Freivalds et al., 1989; Shinohara and Miyano, 1991; Freivalds et al., 1993; Goldman and Kearns, 1995; Zhu et al., 2018). Even if we adopt a preference-based order on concepts (Gao et al., 2016), the maximum is still unbounded (it is only bounded on expectation, with strong sampling priors) and incomputable (Hernandez-Orallo and Telle, 2018).

Jan Arne Telle
Department of Informatics, University of Bergen, Norway. E-mail: Jan.Arne.Telle@uib.no

José Hernández-Orallo · Cèsar Ferri
DSIC, Universitat Politècnica de València, Spain. E-mail: {jorallo,cferri}@dsic.upv.es

In this paper we consider the teaching of universal languages with the goal of finding *small* teaching witnesses. The main insight comes when we realize that some concepts could be taught with very few examples, having low teaching dimension, but those examples could be enormously large. Not only does this deviate from any intuitive notion of 'small teaching witness', but it also ignores the prohibitive time needed for the teacher to find such a witness set and the time needed for the learner to identify the right concept, as this time depends on the size of the witness set. The teacher algorithm may not even terminate. As a result of all this, even when the teaching dimension is low, it can still be incomputable in general, and not very representative of how difficult teaching a concept is. In this paper we address this situation, and ask: *can we define a new teaching metric that is more reasonably related to how easy it is to teach a concept, and use this to teach at least a substantial part of a universal language so that both teacher and learner become computable?*

We will show a solution that answers this question in the positive, and describe experiments with an implementation for a specific universal language. Our main result is Theorem 2, showing that we can teach all total functions computable within some fixed time bound using the same learning prior. Several ingredients are needed, but all revolve around the *size* –the encoding length in bits– of the witness set, rather than its cardinality.

Firstly, we consider as *learning prior* a preference order parameterized by a time complexity function that gives preference to small programs that within the time limit have the desired behavior. Because we include the runtime in this order, this preference depends on the witness set. As a result, the teacher can find these small witness sets by using a search guided by a preference metric, derived –in three different ways— from program length and runtime, generating what we call the "teaching book". This setting leads to a teaching procedure where both teacher and learner are computable. The crucial idea is that we enumerate finite witnesses in order of non-decreasing teaching size, as for each size we have a finite number of them. Note this is not possible for teaching dimension (already for teaching dimension 1 the set of witnesses is infinite). The teaching size, defined as the size of the shortest witness set needed to identify the concept, is still unbounded in general.

Secondly, in order to obtain a bounded teaching size *on expectation*, we define a sampling prior, which is simply based on the "teaching book", in such a way that those concepts that appear early in the book have higher probability to be taught. In other words, the sampling prior gives strong preference to those concepts whose simplest compatible program can be taught with a witness set whose total size is small.

Not only does this make things computable and provides ways of having bounded teaching size on expectation but it is also more meaningful. Furthermore, it has a more profound connotation for the notion of teaching in general. For concept languages where examples are structured and can be of any size (e.g., strings), saying that the teaching dimension is 1 means that there is a successful witness set with one example that will identify the concept, but this example can still be huge in terms of size and may be very difficult to find. For instance, a concept that maps every input to the same very long output could still have a small teaching dimension, but the teaching size would be large. With the notion of teaching size, a concept that requires very large examples to be identified would not have a small teaching size, and would then be a concept that is hard to teach.

The transition from teaching dimension to teaching size follows a long tradition in the context of machine learning and machine teaching (from language identification to compression) that we will properly analyze in section 7. Several key challenges in machine learning and machine teaching (e.g., compositionality and structure) could be re-understood under the notion of teaching size. In light of this historical context and the state of the art, the main contributions of this paper are:

- We prove theoretically and show experimentally that there are concepts with small teaching dimension (even 1) and arbitrarily large teaching size.
- Under our new setting based on building a teaching book ordered by the size of the witness set, both learner and teacher are shown to be computable for universal languages for any fixed time bound, something proved impossible for other machine teaching settings.
- To our knowledge, this paper is the first one where a teaching setting is analyzed experimentally for a universal language, showing that interesting programs (e.g., reverse, mirror, shifts, etc.) are teachable with short witness sets.
- More fundamentally, we see experimentally that the size of the teaching set is usually smaller than the shortest expression of the concept to be taught, which is an illuminating justification of why machine teaching from examples makes sense at all.

The rest of the paper is organized as follows. In the next section we formally define the teaching framework using the notion of teaching size in contrast to teaching dimension. We instantiate the framework through three different preference functions: by length (related to a prior based on Kolmogorov complexity K), by length and runtime (related to a prior based on Levin complexity Kt), and finally by length and a time complexity function (related to a prior based on time-bounded Kolmogorov complexity) where we are able to show the most positive results. In Section 5 we discuss further aspects of the sampling bias. In Section 6 we describe an experimental validation for the universal string manipulation language P3, comparing the use of teaching size (allowing an open number of elements in the witness) to one more related to teaching dimension (where the number of elements in the witness is limited). We put the paper in the context of related work in Section 7 and we close the paper with a discussion of the results and their implications.

2 The General Framework

We consider the following setting for machine teaching. We have an infinite example (or instance) space X and an infinite concept class C consisting of concepts that are a subset of X (this subset forms the positive examples for the concept). The goal is that for any concept $c \in C$ the teacher must find a small witness set $w \subseteq X$ of positive examples from which the learner is able to uniquely identify the concept. In our most general setting the examples will be pairs of strings and the concepts will be the outcomes of programs of a Turing-complete language L, also called “universal”, mapping strings to strings.

Every program written in such a language computes some partial function from inputs to outputs, which is undefined on exactly those inputs on which it does not halt. We call two programs equivalent if they compute the same partial function. We consider the L-concepts defined by the language L to be the set of all partial functions computed by programs written in this language, thus establishing a bijection between L-concepts and equivalence classes of programs. Note that in standard terminology a computable function is a total function computed by some program, which by definition must always halt.

The example space will consist of pairs $\langle i, o \rangle$ over the binary alphabet, so-called input/output pairs, in addition to the pair $\langle i, \perp \rangle$. Given a binary string i as input a program p will either go into an infinite loop, denoted by $p(i) = \perp$, or compute a binary string o as output, which we denote by $p(i) = o$. Thus, the program p computes a function $f_p : \{0,1\}^* \rightarrow \{0,1\}^* \cup \perp$ and belongs to the equivalence class of programs compatible with the L-concept c defined by the partial function computed by p. A witness w for the concept c should be a finite subset of its positive example set, that allows the learner to identify a program p compatible with c. Note that we consider only positive examples when identifying programs, always specifying the desired behavior for a given input. When programs can run unbounded, we will never use $\perp$ as an output, because it would be useless for the learner (it cannot in general check that a program goes into an infinite loop). However, we will use pairs of the form $\langle i, \perp\rangle$ in time-bounded scenarios, where teacher and learner can use this as useful information for determining that a program does not stop in the time budget.

An example set $S = \{\langle i_1, o_1\rangle, ..., \langle i_k, o_k\rangle\}$ is just a finite set of binary i/o pairs, used as witness. The size (number of symbols) of a string s (either input or output) is denoted by $|s|$. Since we will be focusing on the size of example sets we need to carefully consider how to encode it. To encode an example set we need to distinguish each string in an i/o pair so that, e.g., $\{\langle 0, 1\rangle, \langle 00, 11\rangle\}$ is not confused with, e.g., $\{\langle 01, 0\rangle, \langle 01, 1\rangle\}$. We can do this either by using delimiters between strings (as we have done in the previous sentence with the brackets, or with a special symbol $0|1|00|11|$, or by using a prefix code that explicitly allows to check when the end of a string has been reached (e.g., 010001010110001111, with Elias coding (Elias, 1975)). We choose some such encoding and let δ be a function from example sets to natural numbers so that $\delta(S)$ is the number of bits needed to encode the example set S under the chosen encoding (in the previous case $\delta(\{\langle 0, 1\rangle, \langle 00, 11\rangle\})$=18).

We say that a concept c satisfies S, denoted by $c \models S$, if $c(i) = o$ for all the pairs $\langle i, o\rangle$ in S. All concepts satisfy the empty set. Just as for concepts, we say that a program p satisfies the example set S if $p \models S$, i.e. it has the i/o-behavior specified by all i/o pairs in S. The equivalence class of programs compatible with concept c is $Class_L(c) = \{p : \forall S, p \models S \iff c \models S\}$.

2.1 Teaching size

Given this, the teaching size (TS) of a concept c could in a first attempt be defined as follows:

$$TS(c) = \min_S \{\delta(S) : \{c\} = \{c' \in C : c' \models S\}\}$$

i.e., as the size of the smallest (using the δ encoding) example set S that allows a learner to uniquely identify c. This minimal set S is known as a witness set for the concept c and the intention is that a learner should be able to use it to infer a program in $Class_L(c)$. Note that the TS depends on both a concept and a learner. Sometimes in the machine teaching literature, it is assumed that an optimal learner for a language is used, so the definition depends on the language and the learner is hence omitted. In this paper the definition of TS depends on a learner that is specified to follow a precise procedure Φ mapping sets to concepts: $\Phi(S) = c$.

However, as we are dealing with a universal language it is clear that witness sets of finite size will not do. This is the case because for any finite witness set there will be an infinite set of distinct concepts satisfying this witness, since on inputs not specified in the witness there is an infinity of different outputs that could be given (e.g., the witness set $\{\langle 0, 1\rangle, \langle 00, 11\rangle\}$ is consistent with a program that doubles the input but also with one that copies the last bit, and infinitely many other computable concepts). Thus we need a learning prior, for example a preference-based total ordering of programs, that will allow the learner to prefer some programs, and thus also some concepts, over others.

Since the concepts and programs form infinite discrete sets a natural idea is to use orderings related to the resources or complexity required by the programs, with the ‘easy’ programs coming first. Consider a universal language L formed by a finite set of instructions where each instruction υ_i is defined by a symbol of an alphabet Υ. We assume a lexicographic order for the symbols in Υ. A program p is defined as a sequence of instructions and its length is denoted by $\ell(p)$. The number of programs of a given length l is $|\Upsilon|^l$ and hence the number of programs of length $\leq l$ is $\frac{|\Upsilon|}{|\Upsilon|-1}(|\Upsilon|^n - 1)$.

Let $\prec$ be a total order of programs, where programs are ordered by length, thus shorter programs with fewer instructions preceding longer programs, and lexicographically for programs having the same number of instructions. When we refer to an ordering of programs by length then the total order given by $\prec$ is always the order we mean, and similarly 'the smallest program' satisfying some constraint is the earliest such program in this order.

We will in the next sections use the $\prec$ ordering of programs to define three distinct notions of teaching size of concepts. We will also discuss, and prove results about, the effectiveness of teaching under these scenarios.

3 Teaching size according to program length preference: $\boldsymbol{TS_\ell}$

We can define the first program for a concept according to ℓ and $\prec$ as follows:

$$\Phi_\ell(c) = \underset{p}{\operatorname{argmin}}^{\prec}\{\ell(p) : p \in Class_L(c)\}$$

where $\operatorname{argmin}_p^{\prec}$ returns the program p that minimises the expression but comes first in the order $\prec$ in case of ties. Clearly, we can simply define the Kolmogorov complexity of a concept as $K(c) = \ell(\Phi_\ell(c))$, i.e., the length of the smallest program computing c.

The most obvious way to apply the total order $\prec$ induced by ℓ to teaching is to require that the learner, given a witness, can distinguish the desired program/concept from all programs/concepts earlier in the order, but not those later. This is done by defining the first program for an example set S according to ℓ and $\prec$ as follows:

$$\Phi_\ell(S) = \underset{p}{\operatorname{argmin}}^{\prec}\{\ell(p) : p \models S\}$$

and similarly $K(S) = \ell(\Phi_\ell(S))$, i.e., the length of the shortest program having the i/o-behavior specified by S. Now we can give the definition of teaching size of a concept c based on ℓ:

$$TS_\ell(c) = \min_S\{\delta(S) : \Phi_\ell(S) \in Class_L(c)\}$$

i.e., the teaching size for a concept c using the order given by ℓ and $\prec$ is the size of the smallest witness set S, under δ encoding, such that the first program in the order that satisfies S is in the equivalence class of c.[1]

If all programs were guaranteed to halt on all inputs then these definitions could be useful. Given S the learner would try programs in increasing size-lexicographic order, i.e. following $\prec$, and the first one having the i/o-behavior specified by S would be the desired $\Phi(S)$, what the learner should find.

[1] We can also define this teaching size using $\prec$ directly, to highlight the similarity with the preference-based teaching dimension (which would minimize over number of i/o-pairs in S rather than $\delta(S)$):

$$TS_\ell(c) = \min_S\{\delta(S) : p \models S \wedge (p' \prec p \Rightarrow p' \not\models S) \wedge p \in Class_L(c)\}$$

However, we are dealing with universal languages and in this case finding the shortest program, i.e., computing the Kolmogorov complexity, is undecidable, so both learner and teacher would in general also be undecidable. A common approach to get computability is to use an approximation of Kolmogorov complexity. Levin's Kt has some interesting properties as the learner would combine program length and runtime. We explore this possibility in Appendix A and we find that for any finite example set S the learner is computable and is able to identify a program with this behavior. However, there will be concepts that cannot be taught. We need another approach.

4 Teaching size according to time-bounded program length preference: TS_ℓ^f

We will now use a time complexity function $f : \mathbb{N} \to \mathbb{N}$ that will restrict how long the learner, and teacher, will run a program before aborting it. We now allow $\langle i, \perp \rangle \in S$. We define $| \perp | = 1$ and for ease of encoding we partition the pairs into $S = S_\perp \cup S_{out}$ with the former containing exclusively the pairs of the form $\langle i, \perp \rangle$ and the latter containing the pairs for which the program must finish in the time budget. The teacher will be filling what we call an f-Teaching Book, a list of *entries* in the form of program/witness pairs (p, w), with p the smallest program f-compatible with w, and w the smallest witness for which p is f-compatible. The teacher will fill the f-Teaching Book by a nested loop, generating example sets in some order of non-decreasing size, breaking ties in a deterministic manner, and for each example set trying programs of increasing lengths until finding the first one with the correct behavior.

First, for computability of the teacher we must ensure that for every example set $S = \{\langle i_1, o_1 \rangle, ..., \langle i_k, o_k \rangle\}$ there will exist some program p_S, a look-up table hard-coded for S, that within the allotted time has the behavior specified by S. Note we only consider example sets that do not contain any contradictory pair (two distinct behaviors on the same input). In most high-level languages p_S can be implemented as a program that hard-codes a trie data structure for S, with inputs in S as nodes of a binary tree whose edges are labelled 0 or 1 such that all descendants of a node have as common prefix the binary string on the path from the root to that node. The empty output will be given on inputs not in S. On a binary input i this program searches for i in the trie and if found, say $i = i_j$, then it outputs o_j unless $o_j =\perp$ in which case it enters an infinite loop. The program p_S can implement the trie as a hard-coded conditional flow structure (tree of if-then-elses) or can be composed of a first part consisting of instructions that based on the input i traverses a trie that comes as a hard-coded data structure in a second part of the program, used as working memory with random access. Assuming some hard-coding of this kind, p_S will have runtime upper bounded by $c \cdot \min\{|i|, i_{max}\} + o_{max}$ where c is a fixed constant and i_{max}, o_{max} are the lengths of the longest input-string and output-string in S. For some settings and some 'low-level' universal languages this time-bound will not be met, but nevertheless there will be some hard-coded program for any example set S. In general, for a language L we let $\lambda_L(n, S)$ be an upper bound on the runtime on any input i of $|i| = n$ alphabet symbols of some fixed, and fast, hard-coded program for S. When L is clear from context we will drop the subscript.

Fixing a universal language L we say that a program p is f-compatible with example set S, denoted $p \models_f S$, if for all pairs $\langle i, o \rangle \in S$ the program p on input i will within $\max\{f(|i|), \lambda_L(|i|, S)\}$ time steps have the specified behavior: if $o \neq\perp$ then it halts and outputs o, and if $o =\perp$ then it does not halt within that time limit. Note that for any L, f and S there is a program f-compatible with S, since the $\lambda_L(|i|, S)$ time bound allows some hard-coded program for S. We use λ because we want all possible witness sets to identify a program in the teaching book. However, the hard-coded programs will be identified only by witness sets that cannot be compressed.

Clearly, using larger time complexity functions we will be able to correctly discriminate between more programs. For a time complexity function f, we define the first program under this new scheme as follows

$$\Phi_\ell^f(S) = \underset{p}{\operatorname{argmin}}^{\prec}\{\ell(p) : p \models_f S\}$$

and the time-bounded Kolmogorov complexity $K^f(S) = \ell(\Phi_\ell^f(S))$, i.e., the length of the shortest program f-compatible with S. We then use this to define the K^f-teaching size of a concept c as

$$TS_\ell^f(c) = \min_S\{\delta(S) : \Phi_\ell^f(S) \in Class_L(c)\}$$

if such S exists and otherwise undefined; i.e. the teaching size based on ℓ and a complexity function f for a concept c is the size of the smallest witness set S (using δ encoding) such that the shortest program f-compatible with S is in the equivalence class of c. For some concepts, e.g., those requiring much more time than allowed by f, their teaching size will not be defined.[2]

Note that we may have $p \not\models_f S$ but $p \models S$ (or vice-versa) when the time-bound f does not allow p to run to completion on some input in S. When teaching concept c we want the learner to find a program p such that $p \models S$ whenever $c \models S$, so care must be taken. Nevertheless, we will see that we can define a learner and teacher and pick time-bound functions so that concepts can be taught properly.

Given an example set S, the learner wants to find $\Phi_\ell^f(S)$, the smallest program f-compatible with S. This program has some fixed length, and therefore there is a finite number of programs preceding it in the $\prec$ order. The learner is computable: try programs in increasing order, i.e. $\prec$ order, and test if the program is f-compatible with S, with the first such program being the answer.

Let us now turn to the teacher. The teacher is computable, by filling what we call an f-Teaching Book, a list of *entries* in the form of program/witness pairs (p, w), with p the smallest program f-compatible with w, and w the smallest witness for which p is f-compatible. A brute-force algorithm to fill the f-Teaching Book, initially empty, is as follows: try all example sets S with no contradictory pairs but allowing $\langle i, \bot\rangle$ pairs, in order of non-decreasing δ encoding size (breaking ties in a deterministic manner) and test if the program p found by the learner on this S (using the strategy described above for the learner) is already in the f-Teaching Book, and if it is not then add the pair (p, S) to the f-Teaching Book.

Observe that for some L-concept c we cannot rule out the possibility that the f-Teaching Book may contain two programs, with two different witnesses, with both programs equivalent to c. For this to happen, the earlier program must time out on one of the i/o-pairs of the witness identifying the later program, among other things. The teaching size of the concept under ℓ and f is in any case well-defined, as the size of the smallest witness.

We show that, for any language L and any partial or total concept in C_L, i.e. the class of concepts defined by L, there exists a time complexity function f so that using f this concept

[2] This teaching size variant could alternatively be defined using $\prec$ as a preference-based order directly:

$$TS_\ell^f(c) = \min_S\{\delta(S) : p \models_f S \wedge (p' \prec p \Rightarrow p' \not\models_f S) \wedge p \in Class_L(c)\}$$

can be taught. Note we are not able to construct f in a computable way, we give only an existence proof.

Theorem 1 *For any universal language L, and for any concept c in C_L, there is a complexity function f so that the f-Teaching Book will contain some (p, w_c) with $p \in Class_L(c)$ and $TS^f_\ell(c) = \delta(w_c)$.*

Proof Given the language L and a concept c in C_L we take p to be the smallest program equivalent to c and define a time complexity function f and a witness w to show that the teacher will find a pair (p, w) for insertion in the f-Teaching Book.

For partial concepts we allow the witness w to contain i/o-pairs consisting of $\langle i, \perp\rangle$ denoting that the equivalent program must go into an infinite loop on input i. Let p be the smallest program equivalent to c. For any program p' smaller than p, since p' is not equivalent to c, there will exist a conflicting pair, either an input i for which $p(i) = o \neq\perp$ but $p'(i) \neq o$, or with $p(i) =\perp$ and $p'(i) \neq\perp$. Let the finite witness w contain for each p' smaller than p one such conflicting pair, in the first case use $\langle i, o\rangle$ and in the second case use $\langle i, \perp\rangle$. We thus have $w = w_\perp \cup w_{out}$ consisting of q pairs $w_\perp = \{\langle i_1, \perp\rangle, ..., \langle i_q, \perp\rangle\}$ where p goes into infinite loop and with q programs $\{p_1, ..., p_q\}$ for which these are conflicting pairs, thus $p_j(i_j) \neq\perp$ for $1 \leq j \leq q$. We also have k pairs $w_{out} = \{\langle i_{q+1}, o_{q+1}\rangle, ..., \langle i_{q+k}, o_{q+k}\rangle\}$ where the output of p is specified.

We now choose any time complexity function f that obeys that for each $1 \leq j \leq q$ we have $f(|i_j|) \geq \tau(p_j, i_j)$, and also for each $q + 1 \leq j \leq q + k$ we have $f(|i_{q+j}|) \geq \tau(p, i_{q+j})$, where $\tau(p, i)$ is the number of steps executed by program p on input i. This will ensure that, for each $\langle i, \perp\rangle \in w_\perp$ being a conflicting pair for some p', we have f big enough to allow $p'(i)$ to run to completion, and for $\langle i, o\rangle \in w_{out}$ we have f big enough to allow $p(i)$ to run to completion.

As the teacher starts filling the f-Teaching Book going through witnesses of increasing size, consider what happens when the witness $w = w_\perp \cup w_{out}$ we have just described is reached. For any program p' smaller than p we have a conflicting pair $\langle i, o\rangle \in w$, if $\langle i, o\rangle \in w_{out}$ then it will be discovered that p' does not output o within the time bound (either it outputs something else, or it does not halt within the time bound) and thus p' is discarded. If instead the conflicting pair is $\langle i, \perp\rangle \in w_\perp$, then f allows time for $p'(i)$ to run to completion on i and the program p' will be discarded. On the other hand, the program p, which is the smallest equivalent to c, will satisfy all the pairs in w, for a pair $\langle i, o\rangle \in w$ it will within time $f(|i|)$ output the correct thing on input i, while for a pair $\langle i, \perp\rangle \in w$ it will within time $f(|i|)$ not halt on input i.

Thus when the teacher tries the witness w the program p will be found, and if it is not already in the f-Teaching Book it will be inserted. Since the teacher algorithm goes through witnesses in order of increasing size, there will be some witness w_c with smallest size that leads the learner to identify program p. The (p, w_c) pair, with $TS^f_\ell(c) = \delta(w_c)$, will be inserted in the f-Teaching Book, but it is not necessarily equal to the (p, w) pair we have specified in the proof.

Since the concepts in C_L are all partial functions computed by programs written in this language, it includes also functions that are not total, i.e., which are undefined on exactly those inputs on which the program does not halt. Therefore, the existence result we have just shown goes beyond the set of computable functions, i.e., the total functions computed by some program, which by definition must always halt.

In case a concept c in C_L is total, thus defining a computable function, then any equivalent program will halt on every input and the time complexity function f in the above result will depend only on c. This allows us to strengthen the result to teach the whole class of concepts that are computable within the same time bound, and to avoid the non-constructive nature of the previous result. On the other hand, the previous result held for any concept in C_L whereas the result we show now holds only for a subset of the concepts, that we now define. Fix any $t : \mathbb{N} \to \mathbb{N}$ and define the class C_L^t of total concepts in C_L having implementations with runtime $t(n)$:

$$C_L^t = \{c : \exists p \in Class_L(c) \wedge \forall i : \tau(p,i) \leq t(|i|)\}$$

Theorem 2 *For any runtime function t and any universal language L, using time complexity function defined by $f(n) = t(n)$ for all n, we can teach the concept class C_L^t. For any $c \in C_L^t$ the f-Teaching Book will contain some (p,w) with $p \in Class_L(c)$ and $TS_\ell^f(c) = \delta(w)$.*

Proof Consider any concept $c \in C_L^t$ and its equivalent program p_c whose runtime on any input of size n is upper-bounded by $t(n)$. We use an analogous proof as in Theorem 1, to find a witness $w = \{\langle i_1, o_1\rangle, ..., \langle i_k, o_k\rangle\}$ for p_c (rather than for "the smallest program equivalent to c") and note that i) for a smaller program equivalent to p_c there is no conflicting pair, and ii) we will not have conflicting pairs of type $\langle i, \perp\rangle$, thus the constraints on the time complexity function f will be bounded only by the runtime of the program p_c. We define f so that for each $1 \leq j \leq k$ we have $f(|i_j|) = t(|i_j|) \geq \tau(p_c, i_j)$, which allows p_c to run to completion on each of the inputs in w. Thus, for each concept in C_L^t the constraints on $f(n)$ are never higher than $t(n)$. The remainder of the proof proceeds in a manner analogous to the proof of Theorem 1.

Note there is no non-constructivity here, we fix $f = t$ and basically show that for any $c \in C_L^t$ if we let the teacher run long enough then a program equivalent to c will be inserted in the f-Teaching Book. Our setting is not using "coding tricks" (Balbach, 2007; Balbach and Zeugmann, 2009), such as assuming a coding between instances and concepts, where only the "index" would be necessary to identify the concept. Goldman and Mathias's condition holds: whenever a learner identifies a concept c with an example set S, it must also identify c with any superset of S that is also consistent with c (Goldman and Mathias, 1993). This is true for any $c \in C_L^t$ since if the t-time-bounded program $p \in Class_L(c)$ was identified by witness w and w' is a superset of w consistent with c, then since the time bound holds for any input, w' will still identify the same program p. The fact that we remove duplicated programs from the teaching book does not affect this condition. For Theorem 1, however, only a softer version of the condition holds: for any superset of the witness containing extra i/o-pairs for which the program (or an equivalent one) can run inside the time complexity bound.

5 Sampling prior and expected teaching size

With the learning prior ℓ and the time complexity function f seen in the previous section, the algorithm for filling the f-Teaching Book will for any example set S indeed terminate, for the same reason that the learner algorithm terminates, i.e., because of the time limit, the fact that we only run on witnesses S with no contradictory pairs, and we ensure that we allow $\lambda(n, S)$ time on an input of size n, so there is always some program f-compatible with all pairs in S.

Sometimes, especially when the teaching setting is used to understand the complexity of machine *learning*, the concepts to be taught (or its distirbution) are determined by the task. However, in an actual *teaching* task (or in an evaluation setting), it is usually the teacher (or the evaluator) who can decide what order or distribution (especially in an exam) she or he may choose. If a teacher has the task of choosing a convenient sample, there are some constraints influencing this choice. First, both teacher and learner would not like to have a very large teaching or evaluating session. Consequently, giving preference to those concepts with short witness sets makes sense. Second, by choosing a distribution that decays with the order in the book, we ensure that the teacher does not need to prepare a huge book and only enlarge it occasionally. Of course, there are also other cases, such as the teacher following a teaching book sequentially.

We can generalise all these cases, by considering that the teacher has a sampling prior v over all concepts, such that $\sum_{c \in C} v(c) = 1$. With this, we can define the expected teaching size of the whole class C as follows:

$$\mathbb{E}_v[TS^f_\ell(C)] = \sum_{c \in C} v(c) \cdot TS^f_\ell(c)$$

Note that those concepts that are not found in the book will have probability 0. Taking this sampling prior into account the teacher can build a finite f-Teaching Book by setting a limit on the maximum witness size at which f-Teaching Book stops. The teacher can assign decreasing probabilities to the n elements in this finite book, according to their index j in the book, e.g., $v_j = 2^{-j}$, and leave part of the mass of the distribution unassigned for the part of the book that is not calculated ($v_{rest} = 2^{-n+1}$). Whenever the teacher samples from this distribution, if the chosen index $j < n$ then the jth entry (p, w) in the teaching book is chosen and w sent to the learner. Otherwise, the teacher will have to enlarge the book up to the necessary index to generate the example.

Note that in the resulting f-Teaching Book the majority of concepts will usually have teaching size close to the size-limit, as there are more large witnesses. Nevertheless, a sufficient condition to get any desired *expected* finite teaching size derives directly from the equation above. If we take the teaching book, we have the size of the witness set in each entry and we only have to choose a probability distribution such that this sum is lower than 1, with the remainder just expressing the probability that we have to extend the book.

We can also play with various definitions of “interesting” concepts and give higher prior to these, instead of the order that is given by the book. In other words, the teaching book (up to a given size) is an introductory book up to teaching complexity, from which the teacher can assign the desired teaching prior and sample from it (with or without replacement).

6 Experimental validation and insights

We describe an experimental validation of our method for teaching a universal language, and we will derive several insights from the experiments. We work with the universal language P3, a simple language for string manipulation, that we describe below. As learning prior we take the preference order $\prec$ built on program size and complexity function f. We will thus be computing the teaching size TS^f_ℓ of P3-concepts, i.e. concepts computable by P3 programs. We implement the teacher algorithm described in Section 4.

For the sake of simplicity we use a naive encoding δ of example sets based on three delimiters, inserted before every alphabet symbol, to show either that a new i/o string starts or the string continues. This inefficient encoding uses a number of bits that is $\lceil (2 \log_2 5)n \rceil$

where n is the number of alphabet symbols in the example set. Since this is just a multiplicative factor, in what follows, we will use the number of alphabet symbols in the witness for size. For instance, the example set $\{\langle 0100, 00\rangle, \langle 001, \rangle, \langle 00, 00\rangle\}$ has $n = 13$.

The teacher considers example sets by non-decreasing δ size breaking ties in a deterministic way, and for each example set considers programs in $\prec$ order. For each input/output pair (i, o) of the example set S we run $p(i)$. As soon as we discover that p gives the wrong output (note we do not consider $o = \perp$) or meets the time limit we proceed to the next program. Otherwise the first compatible program has been found and the (p, S) pair is inserted into the f-Teaching Book unless p has already been matched with a smaller witness. When all example sets up to a fixed size bound have been tried we stop, thus having computed the initial part of the infinite teaching book consisting of concepts with smallest teaching size. This experiment, denoted by $D_{\geq 1}$, thus computes the teaching size TS^f_ℓ as described above.

For comparison, we also ran an experiment, denoted by $D_{=1}$, in which the teacher considers only example sets consisting of a single i/o pair, generating these by increasing size, thus filling the $D_{=1}$-teaching book with programs having teaching dimension 1 under the preference-based order given by $\prec$ and under the time constraint given by f. In both cases we allowed the same computing time for filling the teaching book, but with varying f, 2000 and 200 respectively, as the lookup-table P3 program for a single example is much faster than the one for several examples. For a fixed number of alphabet symbols there are obviously fewer example sets if we restrict to a single example, thus we filled the teaching books up to two different limits (7 and 9 alphabet symbols). Table 1 shows that $D_{\geq 1}$ nevertheless went through more example sets and ended up with more programs in the book.

Table 1 Teaching size $D_{\geq}1$ and limiting cardinality to 1 $D_{=1}$

Name	**Time *f***	**Max witness size**	**Example sets**	**Programs**
$D_{\geq}1$	2,000	7	17,252	5.062
$D_{=1}$	200	9	9,217	3,227

We start by describing the language P3, then more details of our implementation for teaching size and for the alternative experiment and a comparison of the results.

6.1 The P3 language

P” is a primitive programming language introduced by Corrado Böhm (1964), and was the first GOTO-less imperative language proved Turing-complete, i.e., universal. One of its better known[3] variants has 8 instructions in total. We employ another variant called P3, also universal and having just 7 instructions: `<>+-[]o`.

P3 programs operate on an input/memory tape of cells, by moving a pointer left (instruction <) and right (instruction >), similar to a Turing Machine. We consider P3-programs that take binary inputs and generate binary outputs. There is also the special symbol dot ‘.’ so that the tape alphabet has 3 symbols $\Sigma = \{0, 1, .\}$. The tape is padded with ‘.’ on cells before and after the binary input. The ‘.’ is crucial for loops and halting of programs. There is also an output tape onto which the symbol in the pointer cell can be written (instruction `o`) in a write-only sequential manner, and if a ‘.’ is output the program halts. The tape alphabet has the cyclic ordering (‘0’ ‘1’ ‘.’) and there are two instructions that overwrite the symbol

[3] `https://en.wikipedia.org/wiki/Brainfuck`

in the pointer cell (+ changes it to the next symbol in the ordering, and `-` changes it to the previous one). Thus, applying + to '.' gives '0'. Finally, the bracket instruction `[` loops to the corresponding bracket `]`, i.e., the instructions between the two brackets are executed repeatedly. The condition of the loop is evaluated on both brackets and exits to the instruction following `]` when the content of the pointer cell is '.'.

The number of programs up to size n are exactly: $\sum_{i=0}^{n} 7^i$. However, several optimizations are implemented. For instance, instructions without the output instruction `o` and programs that do not contain a balanced number of brackets are not executed. We use the following lexicographic order on instructions `<>+-[]o` to define the $\prec$ length-lexicographic ordering of P3 programs. Let us consider as an example a P3 program for left shift (e.g., on input 10010 we want output 00101):

```
>[o>]<[<]>o
```

The program starts with the pointer at the first position of the input string and moves the pointer to the second position by the $>$ instruction. Then, the program prints the rest of the characters of the input string using the loop `[o>]`. After that, the pointer is now on the '.' following the last character of the input string, so the pointer is moved one position to the left: $<$, and then to the first '.' on the left of the input string using the loop: `[<]`. Finally, the pointer is moved forward to the first character of the string and it is printed: `>o`.

In the experiments below we can use the example set $\{\langle 0100, 1000\rangle, \langle 10010, 00101\rangle\}$, which has two i/o-pairs, to find this program, as there is no shorter program that has this particular i/o-behavior. However, if we wish to find this program using a single i/o-pair we will not succeed, as the programs `>[o>]+o` and `>[o>]-o` are shorter and one of them will have the behavior of the single i/o-pair, by skipping the first input character, a 0 or 1, printing the rest and then applying + or − to the '.' of the pointer cell to print a 0 or 1. Thus, under preference order $\prec$ the left shift P3 program has teaching dimension two.

6.2 $D_{\geq 1}$: Computing teaching size of P3 programs

In this section we report on an implementation of the teacher and learner algorithm computing teaching size according to time-bounded program length preference: TS_l^f. We thus compute the $D_{\geq 1}$-Teaching Book of P3 programs ordered by teaching size. We choose a constant time limit function $f(n) = 2000$ for all n. This is large enough, given the maximum size of our i/o-pairs that the lookup-table programs can run to completion, and for the course of the experiment we believe the time limit is only reached by non-halting programs. Table 2 gives some (p, w) pairs, of programs and example sets, that the teacher will find.

Table 2 Sample of pairs of programs and witnesses (p, w) found by the teacher.

Example set	**Program**	**Description**
$\{\langle 0, 0\rangle, \langle 10, 10\rangle$	`[o>]`	identity
$\{\langle 010000, 000010\rangle, \langle 1000, 0001\rangle\}$	`[>]+[<o]`	reverse
$\{\langle 011, 11\rangle, \langle 10001, 11\rangle\}$	`[-[+o+]>]`	filter 0
$\{\langle 011, 0\rangle, \langle 10001, 000\rangle\}$	`[+[-o-]>]`	filter 1
$\{\langle 01, 10\rangle, \langle 0011, 1100\rangle\}$	`[+[o>+]+o]`	swap 1 and 0
$\{\langle 01, 11\rangle, \langle 0011, 1111\rangle\}$	`[[+]-o>]`	convert 0 to 1
$\{\langle 01, 00\rangle, \langle 0011, 0000\rangle\}$	`[[+]+o>]`	convert 1 to 0
$\{\langle 0100, 00\rangle, \langle 001, \rangle, \langle 00, 00\rangle\}$	`[+>]<[-o<]`	remove before last 1
$\{\langle 0100, 1000\rangle, \langle 10010, 00101\rangle\}$	`>[o>]<[<]>o`	left shift

In our experiments, we fill a teaching book up to the size of at most 7 alphabet characters in the example set. We got a total of 111,569 example sets. From this set we removed those example sets that are contradictory, those with repeated pairs, and those where all the outputs are empty, since they identify the empty program. This way 17,252 example sets remain.

For each of the 17,252 example sets, a P3 program was found, with some programs identified by many example sets. The final $D_{\geq 1}$-Teaching Book contains 5,062 distinct programs and their witnesses, i.e., the smallest example set that identifies them. In Table 3, we show some witnesses and their corresponding P3 program in the teaching book.

Table 3 Some programs and witnesses belonging to the teaching book.

Witness	**Program**
$\{\langle 010, 010\rangle\}$	`[o>]`
$\{\langle , 0\rangle, \langle 0, \rangle\}$	`-[-o]`
$\{\langle , 111\rangle, \langle 1, \rangle\}$	`+[+ooo]`
$\{\langle 101, 0011\rangle\}$	`[>oo]`

Of the 5,062 programs in the $D_{\geq 1}$-teaching book 663 have a witness with a single i/o-pair, 2,563 with two i/o pairs, 1,689 with three i/o pairs and 147 with four i/o pairs. In Table 4 we see the frequency of programs in the $D_{\geq 1}$-teaching book according to the number of alphabet symbols in the witness set.

Table 4 Number of programs and number of alphabet symbols in experiment $D_{\geq 1}$.

# Symbols	1	2	3	4	5	6	7
# Programs	2	7	26	107	331	1143	3446

6.3 Teaching size compared to teaching dimension 1

Let us now describe a second experiment, denoted by $D_{=1}$, in which the teacher considers only example sets consisting of a single i/o pair, generating these by increasing size, thus filling a $D_{=1}$-Teaching Book with programs having teaching dimension 1 under the preference-based order given by $\prec$ and time complexity function f. Let us first show that if we do not halt this process it will continue filling the teaching book indefinitely, since there is an infinity of distinct concepts with teaching dimension one.

Lemma 1 *For any integer n, the $D_{=1}$-Teaching Book, will eventually include a program of teaching size at least n.*

Proof By contradiction. Suppose not, it means the $D_{=1}$-Teaching Book would not grow beyond some k programs. But then, say on the empty input ϵ, these k programs can only output at most k distinct outputs. If we take some output x not among these k then there is a smallest program p that on empty input hard-codes the output x. This program would in fact be added to the $D_{=1}$-Teaching Book when running on example set (ϵ, x), since the hard-coded program is allowed to run to completion regardless of the choice of f, contradicting that $D_{=1}$-Teaching Book had only k programs.

For the experiment we choose a constant time limit function $f(n) = 200$ for all n, since the lookup-table program of a single fixed i/o-pair, for the sizes we consider, will not need more than this number of steps. We limit the size of witnesses to at most 9 alphabet symbols. For each binary string of size k and note that this string can be broken up to form a single i/o-pair in $k + 1$ distinct ways. Thus the number of example sets up to size n is in this case given by $\sum_{0 \leq k \leq n}(k + 1)2^k$ and for our limit of $n = 9$ this gives 9,217 example sets. For all these example sets a program was found and the final $D_{=1}$-Teaching Book contained 3,227 distinct programs and their witnesses. Table 5 shows some programs and witnesses in the $D_{=1}$-Teaching Book.

Table 5 Some program and witness pairs in the teaching book of experiment $D_{=1}$.

$\langle$**Input Output**$\rangle$	**Program**
$\{\langle 010, 010 \rangle\}$	`[o>]`
$\{\langle 00, 110000 \rangle\}$	`>>-[oo<]`
$\{\langle , 0010101 \rangle\}$	`>>>+[o<+o+]`
$\{\langle , 11110000 \rangle\}$	`-[oooo-]`

For this experiment we have some observations on the correlation between the size of the input and output of the witness set and the length of the program. First, the length of the programs is highly correlated with the size of outputs and inversely correlated with the size of inputs. When we have short inputs the identified program usually hard-codes the output for that particular example. Given an output of size k, we will need at worst 2 $\times k$ instructions to obtain the desired output: one instruction to get the correct value in the pointer cell (+ or $-$), and the $\circ$ instruction. The length of the programs seems to follow a normal distribution. The median is 9 instructions. If we compare the frequency of example sets identifying a program with respect to size, we see that short programs are identified by more example sets, this is also expressed by the negative correlation of these two variables.

Let us compare witness size and program length for the two teaching books. In Figure 1 we see two scatter plots giving this information. It is clear that the programs in the $D_{\geq 1}$-teaching book, even though there are more of them (for visibility the scales are different), are generally smaller. The $D_{=1}$-teaching book contains many (long) hard-coded programs, since hard-coding is more prevalent when having to satisfy only a single i/o-pair. Also, we notice in both cases a correlation between program length and witness size (which may become tighter if we calculate the books for larger witness sets), and that the bulk of the witnesses, unsurprisingly as there are more larger example sets, are towards the high end.

There were 5,062 programs in $D_{\geq 1}$ and 3,227 in $D_{=1}$, with 543 programs being in common to both teaching books. For each of these programs, we want to compare the size of their witness set in the two teaching books. In Figure 2 we include a scatter plot where each circle represents a group of these 543 programs, sharing the same pair of sizes of witness encodings. The size of the circle represent the number of programs in the group. The x-axis represents the size of the witness set in the $D_{=1}$-Teaching Book, while the y-axis represents the size of the witness set in the $D_{\geq 1}$-teaching book. As expected, no programs are above the diagonal, as the $D_{\geq 1}$-teaching book is built to minimize size.

For programs on the diagonal the two sizes are identical, but there are some exceptions (43 programs). In these cases the size in the $D_{\geq 1}$-teaching book is smaller, see Table 6. For example, the program `+[o>]-o` which is basically "increase cell, loop outputting and move right while 0 or 1, decrease the final cell (which must be a '.') and output" has the $D_{=1}$ witness $\{\langle 0000, 10001 \rangle\}$ and the $D_{\geq 1}$ witness $\{\langle , 10 \rangle, \langle 1, 1 \rangle\}$. Size goes from 9 to 4.

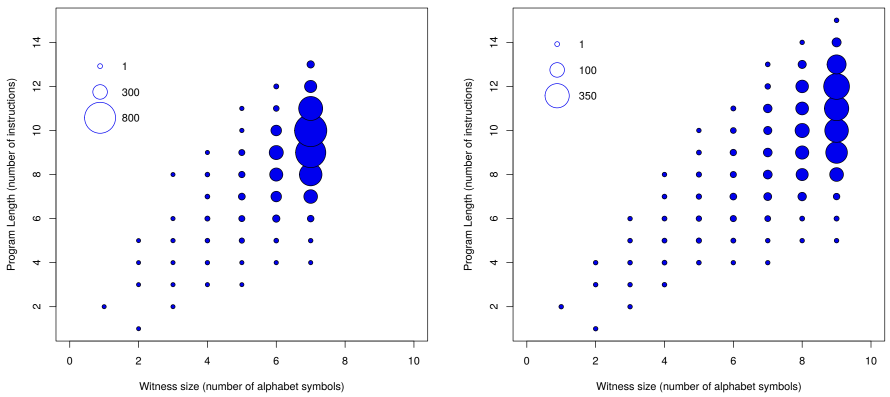

Fig. 1 Scatter plot of programs according to their witness size, in the $D_{\geq 1}$-teaching book (left, witness cardinality greater than or equal to 1) and the $D_{=1}$-teaching book (right, witness cardinality equal to 1). Sizes of the circles are proportional to the number of entries.

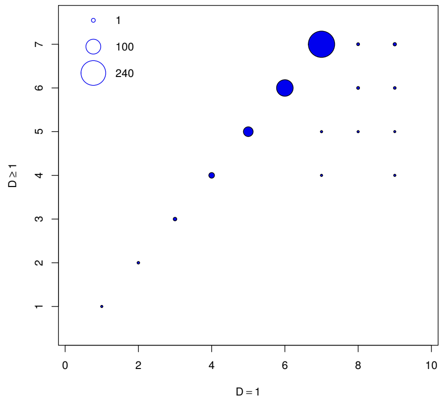

Fig. 2 Comparing size of witnesses in experiment $D_{=1}$ (x-axis) and experiment $D_{\geq 1}$ (y-axis). Sizes of the circles are proportional to the number of entries.

Table 6 Some programs in both teaching books where the witness size is smaller in $D_{\geq 1}$ than in $D_{=1}$.

Program	$D_{\geq 1}$	Size	$D_{=1}$	Size
`+[o>]-o`	$\{\langle, 01\rangle, \langle 1, 1\rangle\}$	4	$\{\langle 0000, 10001\rangle\}$	9
`-[ooo>]+o`	$\{\langle, 1110\rangle, \langle 0, 0\rangle\}$	6	$\{\langle 11, 0001110\rangle\}$	9
`-[o>]+o`	$\{\langle, 10)\rangle, \langle 0, 0\rangle\}$	4	$\{\langle 1001, 00010\rangle\}$	9

6.4 Further experimental results and discussion

Note that for the previous experiments we used a very inefficient encoding of example sets. One of the interesting things about the teaching size is that it gives a quantification of how many bits would be needed to send a witness set that is sufficient for the learner to identify the concept. This means that the efficiency of the encoding is key. Consequently, as an alternative to the simple encoding we used in the previous sections, we ran an experiment using an asymptotically optimal universal code, the Elias Delta Coding (Elias, 1975), which encodes a number x using $\lfloor \log_2(x) \rfloor + 2\lfloor \log_2(\lfloor \log_2(x) \rfloor + 1) \rfloor + 1$ bits. The number of bits depends only on the length of the binary encoding of x and we specify this as a function $ED : \mathbb{N} \rightarrow \mathbb{N}$ so that the Elias Delta code for a binary string of n bits uses $ED(n)$ bits. For an example set $S = \{\langle i_1, o_1 \rangle, ..., \langle i_k, o_k \rangle\}$ its Elias Delta encoding size is $\delta'(S) = \sum_{1 \leq j \leq k} ED(|i_j|) + ED(|o_j|)$, basically concatenating all the encodings. Since these are prefix codes their concatenation preserves the separation into the original binary strings.

We constructed the teaching book enumerating examples sets up to a size of 15 bits under this δ'. Figure 3 compares the witness size with this new encoding with the program length, also expressed in bits (as we have 7 instructions, we multiply the number of instruction of a program by $\log_2 7$ to have its length in bits too).

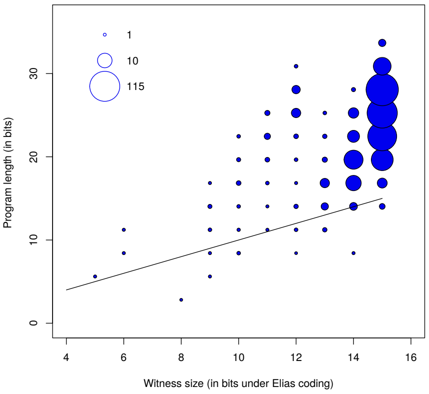

Fig. 3 Scatter plot of program length in bits versus witness size in bits, for the new teaching book using Elias coding. Size of circles proportional to number of programs. The straight line is the unit diagonal.

We see a few small programs that require large witness sets (under the unit diagonal on Figure 3). But the striking finding is that in most other cases – almost always – the witness size (in bits) is smaller than the program size (in bits). This extraordinary observation means that if a teacher wants to send (teach) a concept, it is frequently more efficient (in terms of transmission) to send its optimal witness set under our schema than to send the program itself. This is significant experimental support for the phenomenon (usually observed in humans, Khan et al. 2011) that sometimes it is more information-efficient to teach some ideas using examples than explaining the idea itself. The very problem of teaching by example –"if we already know the true model, why bother training a learner?" in (Zhu, 2015)'s words– is ultimately meaningful under these findings, and goes beyond the traditional applications for education and adversarial attacks. It is not that the teacher "cannot telepathize [the model]" Zhu et al. (2018), it is that showing a few examples instead could be more efficient. The fact that we can now compare the magnitudes of programs and witnesses in the same unit (bits) is given by the notion of teaching size, rather than teaching dimension.

Now for this new encoding, we are going to calculate the expected teaching size using some different sample priors. For instance, if we use the sampling prior as defined in section 5 (let us call it v_1), we get an expected teaching size of 5.28. This is clearly a very extreme prior so we get very close to 5 bits, which is the minimum witness size for Elias coding. A less extreme sampling prior v_2 would be to assign a uniform distribution on each size bin up to the maximum size in the book (15). That gives us a mass of probability $1/(15-5)$ for each size (note that there is a size bin with no entries). With this prior, we get an expected teaching size of 10.2 bits. With a uniform distribution v_3 on all the entries in the book, we would have an expected teaching size between 14 and 15.

In general, we do not know whether a concept appears twice in the book, but if the complexity function f is large enough in terms of the limit up to which the book is constructed (as in this case), redundant programs will be quite unlikely. So, Figure 3 can be interpreted as a comparison between $TS^f_\ell(c)$ and $K^f(c)$, where we see that $TS^f_\ell(c) < K^f(c)$ for most cases[4]. An interesting property to analyze for a pair of language and sample prior v is the transmission efficiency of teaching, defined as $\mathbb{E}_v[K^f(C)] - \mathbb{E}_v[TS^f_\ell(C)]$, where $\mathbb{E}_v[K^f(C)]$ is $\sum_{c\in C} v(c) \cdot K^f(c)$, also measured in bits. For P3, up to the sizes we have explored, this efficiency is positive for the three priors we have used above (v_1, v_2 and v_3).

From the comparison of experiments we have the following observations:

- The computability of the theoretical setting is translated into a universal language, P3.
- Teaching books have a high number and high diversity of programs. Hence we do not observe any collapsing phenomenon of a few programs capturing almost all witness sets.
- We can find interesting programs such as those in Table 2.
- The experiment with limited witness cardinality equal 1 shows higher overfitting than the general setting.
- With a strong sampling bias, we have a very small expected teaching size, but other softer sampling bias can be used.
- Remarkably, it is rather common to find witness sets that are smaller than the programs they identify. This gets more pronounced for efficient encodings such as Elias delta.

All the code for the P3 intepreter, the teacher and the learner, and the several settings and encodings used in this paper, as well as results and plots, can be found at this repository: `https://github.com/ceferra/The-Teaching-Size-with-P3`.

7 Related Work

In this section we put the notion of teaching size in the historical context of machine learning and machine teaching. We start with the early days of machine learning, when several theoretical learning models were introduced. Gold's language identification in the limit (Gold, 1967) can be understood as a first model of teaching, where positive (or negative) examples are given sequentially. The emphasis is on whether the concept that is generating the sequence of examples will be identified. When sequences are replaced by sets we have another model, usually known as concept learning (or teaching). The link between the number of examples needed and the complexity of the concept class has been investigated profusely, most especially under the notions of VC-dimension (Vapnik and Chervonenkis, 1971) and PAC-learnability (Valiant, 1984). In all these models and variants, the magnitude or complexity of the evidence is usually measured in terms of cardinality, i.e., how many examples are necessary.

For many concept languages, the examples do not have structure, so all examples are equally complex. This view is reinforced by a custom setting in machine learning where

[4] Note that it is also the case that $TS^f_\ell(c) > K^f(c) + k$ where k is a constant given by the length of expressing the program "Run the learner algorithm on w" in the language.

examples are just points in a multidimensional space, i.e., a vector of real numbers. On top of this, the dominant paradigm in machine learning today is that of function approximation, rather than function identification. However, when examples have structure, we need concepts that are able to compose elements, and then the notion of approximation does not make sense (e.g., if a hypothesis outputs '11110000' rather than '11110010', the output may look close, but the true concept may still be quite afar). As a result, there is an increasing interest in the understanding of learning where programs or other structural concepts are built or induced from non-scalar data (Lake et al., 2015; Gulwani et al., 2015; Lake and Baroni, 2018; Lázaro-Gredilla et al., 2019).

Alongside the theoretical and practical models of machine learning, there has been signficant progress on the notion of machine teaching, where the teacher can choose the examples at will, the optimal "witness set" (Freivalds et al., 1989; Shinohara and Miyano, 1991; Freivalds et al., 1993; Goldman and Kearns, 1995), in order to make learning more efficient. While this gives extra power (as other variants of learning, such as active learning), the theoretical models have not yet shown a setting where teaching becomes efficient for rich concept classes, not to say universal languages. A remarkable exception is the model of teaching in (Angluin and Kriķis, 2003), where, given the right assumptions, the learner can learn any partial recursive function. The focus is on the ability to learn and not on the minimum cardinality (or size) of the examples.

The most common theoretical model since then is based on the notion of teaching dimension, and appeared together with the early theoretical models of teaching. One way of tightening the choice when many concepts are compatible with a given witness set is by the use of costs or preferences, following the tradition of the K-dimension (Balbach, 2007, 2008) and consolidated under the notion of the preference-based teaching dimension (PBTD) (Gao et al., 2016). Given an order on concepts, we just choose the first compatible concept in this preference order. With the use of preferences we get some finite (worst-case) teaching dimensions for some restricted languages (Gao et al., 2016).

Even these refinements of the teaching dimension fall short for a full underpinning of machine teaching, which is moving towards more general settings (Khan et al., 2011; Zhu, 2013, 2015; Simard et al., 2017), and related areas such as curriculum learning (Bengio et al., 2009), understood as teaching with sequences of examples of increasing complexity or requiring from what has been learned before. This divergence (Khan et al., 2011) also happens in the context of new insights about teaching in cognitive science and psychology (Shafto et al., 2014), where composition and structure require rich concept classes.

Some recent variants of the teaching dimension have paid attention to languages that are structurally rich and universal. Hernandez-Orallo and Telle (2018) studied the teaching dimension when using a learning prior that was derived from a complexity measure. They showed that even rich infinite languages, such as the regular languages, can be taught with an upper bound on the *expected* number of examples in a witness. Two priors are needed for this, the learning prior – basically a variation of the preference-based order (Gao et al., 2016) – used to inform the learner of how to search for concepts given a witness set, and the sampling prior used by the teacher to test the performance of the learner. However, not only are there several incomputability problems, but the teaching dimension does not really capture how complex it is to learn with rich languages producing arbitrarily large examples.

Replacing the teaching dimension by the teaching size and the setting we present in this paper solves these issues and links machine teaching to some fundamental concepts in machine learning such as compression (through Kolmogorov complexity) and Solomonoff's induction theory (1964). The size of the inputs (and outputs) is now placed at the forefront, and how compressible they are play a key role in the understanding of machine teaching in light of the theoretical and experimental results shown in this paper.

While theorem 1 applies for any universal language, it gives a result where a different complexity function can be chosen for each concept. On the contrary, the time bound is used to define the relevant concept class for theorem 2. While this restricts the concept class, it also allows for the analysis of the teaching size for a hierarchy of concept classes according to their time complexity functions. For instance, interesting connections could be established when choosing functions in P and compare the resulting teaching sizes (and book) with respect to various concepts and classes.

Finally, the shift from the teaching dimension to the teaching size, and the use of the sampling bias to calculate the expected teaching size links us to fundamental notions in communication and information theory, and reinforces the view of machine teaching from a coding perspective (Zhu, 2015). In our setting, the learner is actually decoding the examples to find the message (the concept). In the experiments, witness sets can be seen as a compressed version of a concept, assuming that the learner is using a decoding algorithm that searches programs by increasing length. Under this view, what really matters is the comparison of sizes, and not cardinalities, between witness sets and programs.

8 Conclusions

In this paper, we have argued that the concept of teaching dimension is counter-intuitive for languages where examples are structured, and can be arbitrarily long. A small teaching dimension could be obtained with a very long example. Also, the teaching dimension leads to incomputability problems for universal languages. Instead, we have introduced the teaching size, and we have analyzed the computability of the teaching procedure (for learner and teacher) according to length, interwoven length and computational steps, and length with a time complexity function. In the last case, we have been able to show positive computatiblity results, while preserving the power of teaching the whole class in general.

The use of size instead of cardinality also makes the results in expectation for a whole class of concepts more meaningful. For instance, having a finite dimension for a concept class may still mean that the examples may have an unbounded size, and hence the process is not bounded at all. With the use of the teaching size this cannot happen. Actually, instead of the expected teaching size (the mean for the sampling prior), we could also calculate the total teaching size for teaching n concepts (the sum for the sampling prior). Also, the coupling with the complexity function f paves the way for bounding both teaching and learning time on expectation (or on aggregation).

Many of the traditional and new applications of machine teaching can be strengthened under the view of the teaching size paradigm, including human-computer interaction, education, trustworthy AI and coding (Zhu et al., 2018). For instance, interaction with robots and personal assistants can benefit from a better understanding of how humans teach (and communicate) concepts, and bring the old ideas of programming by example (Lieberman, 2001) closer to reality. A user can tell an assistant that she likes peas with roasted chicken and potatoes, but dislikes peas with fish, and the system can infer that this is sufficient to infer the kind of recipes the user likes. A model of teaching where these short examples are put at the same level of examples having millions of ingredients does not make any sense.

The notion of teaching size can have a novel applicability for explaining ML models (Biran and Cotton, 2017), where key examples have to be given to the user. How to find the key *short* examples to convey the meaning of the model is precisely what we have analyzed here. Finally, while we tend to associate structured attributes with lists, graphs and other compositional objects, natural numbers and real numbers can also be considered structured, with an encoding to define their size, using e.g. rationals to encode real numbers. In this light, it is not the same to teach a concept with a few numbers of small size as to teach it with a single number of huge size. This would explain why humans do not use the number that is "closest to the decision boundary" (Khan et al., 2011) at the cost of using infinite precision.

The experimental section supports the intuition that more examples usually pave the way for witnesses of smaller size. Moreover, a focus on minimizing the number of examples leads to specialization (hard-coded programs for one example), while a focus on minimizing size usually finds more general programs, with large numbers of examples being used to identify diversity.

In terms of tractability, the learning algorithm grows exponentially, but it is fully parallelizable. The teaching algorithm uses the learning algorithm for each witness function it iterates on. Even if we have to check whether the concept is already in the book or not, this can be done as a postprocess, so the teaching algorithm is also parallelizable at the upper level. Also note there are very efficient specialised computers (Jun, 2016) for the language BF – very similar to P3 – with which we could produce much larger teaching books.

The most significant insight from the experiments was finding that the size of the program is almost always larger than the optimal witness set used to identify it. This suggests that sending the examples rather than the definition of the concept pays off, a phenomenon we observe in real teaching scenarios with humans. This is possible whenever the learner has a powerful learning mechanism and an aligned learning prior.

The teaching size opens a new avenue for a more realistic and powerful analysis of machine teaching, and suggests closer connections with learnability theory and communication, especially when results are expressed in terms of the size of the inputs, and not only the number of examples. As future work we want to explore several non-universal concept classes and sampling priors to estimate their expected teaching size.

References

Angluin D, Kriķis M (2003) Learning from different teachers. Machine Learning 51(2):137–163

Balbach FJ (2007) Models for algorithmic teaching. PhD thesis, University of Lübeck

Balbach FJ (2008) Measuring teachability using variants of the teaching dimension. Theoretical Computer Science 397(1-3):94–113

Balbach FJ, Zeugmann T (2009) Recent developments in algorithmic teaching. In: Intl Conf on Language and Automata Theory and Applications, Springer, pp 1–18

Bengio Y, Louradour J, Collobert R, Weston J (2009) Curriculum learning. In: Proceedings of the 26th annual international conference on machine learning, ACM, pp 41–48

Biran O, Cotton C (2017) Explanation and justification in machine learning: A survey. In: IJCAI-17 Workshop on Explainable AI (XAI), p 8

Böhm C (1964) On a family of turing machines and the related programming language. ICC Bull 3(3):187–194

Elias P (1975) Universal codeword sets and representations of the integers. IEEE transactions on information theory 21(2):194–203

Freivalds R, Kinber EB, Wiehagen R (1989) Inductive inference from good examples. In: International Workshop on Analogical and Inductive Inference, Springer, pp 1–17

Freivalds R, Kinber EB, Wiehagen R (1993) On the power of inductive inference from good examples. Theoretical Computer Science 110(1):131–144

Gao Z, Ries C, Simon HU, Zilles S (2016) Preference-based teaching. In: Conf. on Learning Theory, pp 971–997

Gold EM (1967) Language identification in the limit. Information and control 10(5):447–474

Goldman SA, Kearns MJ (1995) On the complexity of teaching. J of Computer and System Sciences 50(1):20–31

Goldman SA, Mathias HD (1993) Teaching a smart learner. In: Conf. on Computational learning theory, pp 67–76

Gulwani S, Hernández-Orallo J, Kitzelmann E, Muggleton SH, Schmid U, Zorn B (2015) Inductive programming meets the real world. Comm of the ACM 58(11)

Hernandez-Orallo J, Telle JA (2018) Finite biased teaching with infinite concept classes. arXiv preprint arXiv:180407121 URL `https://arxiv.org/abs/1804.07121`

Jun SW (2016) 50,000,000,000 instructions per second: Design and implementation of a 256-core brainfuck computer. Computer Science and AI Laboratory, MIT

Khan F, Mutlu B, Zhu X (2011) How do humans teach: On curriculum learning and teaching dimension. In: Advances in Neural Information Processing Systems, pp 1449–1457

Lake B, Baroni M (2018) Generalization without systematicity: On the compositional skills of sequence-to-sequence recurrent networks. In: ICML, pp 2879–2888

Lake BM, Salakhutdinov R, Tenenbaum JB (2015) Human-level concept learning through probabilistic program induction. Science 350(6266):1332–1338

Lázaro-Gredilla M, Lin D, Guntupalli JS, George D (2019) Beyond imitation: Zero-shot task transfer on robots by learning concepts as cognitive programs. Science Robotics 4

Levin LA (1973) Universal Search Problems. Problems Inform Transmission 9:265–266

Li M, Vitányi P (2008) An Introduction to Kolmogorov Complexity and its Applications. 3rd Ed. Springer

Lieberman H (2001) Your wish is my command: Programming by example. Morgan Kaufmann

Shafto P, Goodman ND, Griffiths TL (2014) A rational account of pedagogical reasoning: Teaching by, and learning from, examples. Cognitive psychology 71:55–89

Shinohara A, Miyano S (1991) Teachability in computational learning. New Generation Computing 8(4):337–347

Simard PY, Amershi S, Chickering DM, Pelton AE, Ghorashi S, Meek C, Ramos G, Suh J, Verwey J, Wang M, et al. (2017) Machine teaching: A new paradigm for building machine learning systems. arXiv preprint arXiv:170706742

Solomonoff RJ (1964) A formal theory of inductive inference. Part I. Information and control 7(1):1–22

Valiant LG (1984) A theory of the learnable. Comms of the ACM 27(11):1134–1142

Vapnik VN, Chervonenkis AY (1971) On the uniform convergence of relative frequencies of events to their probabilities. Theory of Probability and Its Applications 16:264—280

Zhu X (2013) Machine teaching for bayesian learners in the exponential family. In: Neural Information Processing Systems 26, Curran, pp 1905–1913

Zhu X (2015) Machine teaching: An inverse problem to machine learning and an approach toward optimal education. In: AAAI, pp 4083–4087

Zhu X, Singla A, Zilles S, Rafferty AN (2018) An overview of machine teaching. arXiv preprint arXiv:180105927

A Teaching size according to Levin's search: $TS_{\ell\tau}$

To get a finite procedure we consider the introduction of computational steps in the complexity function, inspired by Levin's Kt (Levin, 1973; Li and Vitányi, 2008). We define the order given by Levin's search for an example set $S = \{\langle i_1, o_1\rangle, ..., \langle i_k, o_k\rangle\}$ as follows:

$$\ell\tau(p, S) = \ell(p) + \log \sum_{\langle i,o\rangle \in S} \tau(p, i)$$

where $\tau(p, i)$ represents the runtime of executing program p on the input i. And for a given S, the first program found under Levin's search is given by:

$$\Phi_{\ell\tau}(S) = \underset{p}{\operatorname{argmin}}^{\prec}\{\ell\tau(p, S) : p \models S\}$$

$\Phi_{\ell\tau}(S)$ is the Levin-simplest program satisfying S. And Levin's Kt is simply $Kt(S) = \ell\tau(\Phi_{\ell\tau}(S))$ Note that $Kt(S)$ is related to the logarithm of the time it takes to generate the behavior S, when we execute all possible programs in dovetail fashion, by increasing $\ell\tau$ (following the program size ℓ with $\prec$ for ties and runtime τ). We define the teaching size of a concept c under $\ell\tau$ as follows:

$$TS_{\ell\tau}(c) = \min_{S}\{\delta(S) : \Phi_{\ell\tau}(S) \in Class_L(c)\}$$

i.e. the size of the smallest witness set S, using δ encoding, such that the Levin-simplest program satisfying S is in the equivalence class of c.

When the learner is given a set of instances $S = \{\langle i_1, o_1\rangle, ..., \langle i_k, o_k\rangle\}$ – note in this case we do not allow a pair $\langle i, \perp\rangle$ – we want to find $\Phi_{\ell\tau}(S)$, the Levin-simplest program satisfying S.

The learner is computable: given a witness we can perform the dovetail procedure of Levin's universal search, on an increasing 2-dimensional budget: over programs of increasing length ℓ following $\prec$ and over increasing runtimes τ, and in this way we find the Levin-simplest program satisfying the witness.

Let us now turn to the teacher. The teacher will be able to distinguish between concepts defined by any finite behavior, by filling what we call a *Levin Teaching Book* of program/witness pairs (p, w) with p the Levin-simplest program satisfying w, and w the smallest witness for p.

The Levin Teaching Book can be computed following a brute-force algorithm. Starting with an initially empty book, try all witnesses w with no contradictory pairs (i.e., no two pairs with distinct outputs for the same input) and no $\langle i, \perp\rangle$ pairs, in order of increasing δ encoding size (lexicographically for those of the same size) and run the learner algorithm (i.e., Levin search) on w to find a program p. Check if p is already in the Levin Teaching Book, and if it is not then add the pair (p, w) at the end of the Levin Teaching Book. In that way, the Levin Teaching Book is formed by pairs (p, w), where w is a witness and p the Levin-simplest program satisfying it. Observe that for some L-concept c we cannot rule out the possibility that the Levin Teaching Book may contain two programs, with two different witnesses, with both programs equivalent to c. This would happen, if for example, one of the programs takes an extraordinary long time on one of the i/o-pairs of the witness of the other program, and vice-versa. The teaching size of the concept under $\ell\tau$ is in any case well-defined, as the size of the smallest witness.

We have the following positive result.

Claim For any universal language L and finite example set S, there will exist some (p, w) with $p = \Phi_{\ell\tau}(S)$ in the Levin Teaching Book. If $p \in Class_L(c)$ then $TS_{\ell\tau}(c) \leq \delta(w) \leq \delta(S)$.

Proof When the teacher tries S then the Levin-simplest program p satisfying S will be found. Either (p, S) will be inserted in the Levin Teaching Book or p has already been found paired with a smaller witness w. If $p \in Class_L(c)$ we have $TS_{\ell\tau}(c) = \delta(w) \leq \delta(S)$ unless there is a program p' equivalent to p that is Levin-simplest for some example set smaller than w.

This means that, if we want to teach not a specific concept, but require only that we teach one out of a number of concepts having a given behavior S on a finite input set, then we can do so. However, we would like, for every universal language, to teach also total concepts with a specific *infinite behavior*. For a simple example, consider teaching the identity concept, whose equivalence class contains exactly the id-programs, i.e., any program that gives as output the same as its input. Without putting some constraint on the universal language we cannot do that, as seen by the following result, formulated using Universal Turing Machines.

Claim For any Universal Turing Machine (universal language) U there exists another UTM U' s.t. no id-program will be in the Levin-Teaching Book for U'.

Proof We give a sketch of the proof. We construct U' so that it alters the programs of U by ensuring 1) the existence of fast non-id programs such that for any finite witness set there is one that behaves like an id-program and 2) any real id-program is slowed down so it is not the Levin-simplest for any finite witness set. We ensure these two conditions as follows: 1) Make a computable set $\{p_i : \text{binary string } i\}$ of programs such that $p_i(x) = x$ except for $p_i(i) \neq i$, and let this be fast. 2) For any other program p on input x add a conditional such that before halting we check if the output is $p(x) = x$ and if so we add some code slowing it down. Then, for any proposed finite witness w for the id-program we have some smallest input string $i \notin w$, and thus the program p_i will be Levin-simpler on w than any of the real id-programs.

Thus, even for quite simple concepts we cannot avoid that there will be some contrived universal languages that 'fool' the teacher using $\ell\tau$ so that a witness set for this concept cannot be found.