On the Computability of Teaching a Universal Language

Cesar Ferri

DSIC, Universitat Politècnica de València, Spain

`cferri@dsic.upv.e`

José Hernández-Orallo

DSIC, Universitat Politècnica de València, Spain

`jorallo@dsic.upv.es`

Jan Arne Telle

Department of Informatics, University of Bergen, Norway

`Jan.Arne.Telle@uib.no`

July 15, 2019

Abstract

The theoretical hardness of machine teaching has usually been analysed 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

Jose: The title doesn't reveal too much, which is good and bad at the same time. It doesn't emphasise the teaching size. What about something around these lines: "The Teaching Size: Computable teachers and learners for universal languages"

JA: Sure

Turing-complete language, showing how teaching dimension and teaching size differ in practice.

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.

Even if we adopt a preference-based order on concepts, such as in the preference-based teaching dimension [3], the maximum is still unbounded (it's only bounded on expectation, with strong sampling priors) and incomputable [4].

Jose: Here I have removed material about our previous work. It could go to a related work section, if we decide to add one.

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 is still uncomputable in general, and not very representative about 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

Jose: I've rewritten these two paragraphs extensively. Please check.

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 double dovetail search –on size of witness set and a Levin search for each of them, generating what we call the “teaching book”. This setting leads to a teaching procedure where both teacher and learner are computable. To get computability this is crucial, since we can 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, since for teaching dimension 1 the set of witnesses is already 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. There is no upper bound on the teaching size of concepts of teaching dimension 1; consider, e.g., an infinite set of programs each giving a distinct output on empty input. 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 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 in 3 separate sections: by Kolmogorov complexity K, Levin complexity Kt, and finally by time-bounded Kolmogorov complexity K^f where we are able to show the most positive results. In Section 6 we discuss further aspects of the sampling bias. In Section 7 we describe an experimental validation for the universal string manipulation language P3, comparing two implementations focused on teaching size and teaching dimension. We end 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. In fact, some of our results will hold for all L-concepts and thus go beyond the set of computable functions.

The example space will consist of pairs $< i, o >$ $\leftarrow\!\bullet$ $\leftarrow\!\bullet$ over the binary alphabet, so-called input/output pairs, in addition to the pair $< i, \perp >$. 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 $p : \{0,1\}^* \to \{0,1\}^* \cup \perp$ 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. In some cases, since we cannot in general check that a program goes into an infinite loop, we will not allow a witness or example set to contain the pair $< i, \perp >$. However, when we consider time-bounded behaviors we will allow this as well.

$\leftarrow\!\bullet$ Jose: Can we use $\langle \ldots \rangle$ instead of $< \ldots >$?

$\leftarrow\!\bullet$ JA: Sure, although it is an ordered pair

An example set $S = \{< i_1, o_1 >, ..., < i_k, o_k >\}$ is just a finite set of binary i/o pairs, and these will be used as witnesses. To encode example sets, we view each binary input-string (and output-string) as a number x and use the Elias

Delta Coding [2] that encodes 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 $ED : \mathbb{N} \to \mathbb{N}$ so that the Elias Delta code for a binary string of n bits uses $ED(n)$ bits. For an example set $S = \{< i_1, o_1 >, ..., < i_k, o_k >\}$ 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 say that a concept c satisfies S, denoted by $c \models S$, if $c(i) = o$ for all the pairs $< i, o >$ 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 $Class(c) = \{p : \forall S, p \models S \iff c \models S\}$.

←● Jose: I've moved these two sentences here. The rest remains the same

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 Elias encoding) example set S that uniquely identifies c. This minimal set S is known as a witness set for the concept c. Given a witness set S for a concept c the learner must be able to infer a program in $Class(c)$.

However, as we are dealing with a universal language it is clear that with this definition the witness sets would be infinite. This since for any finite witness set there will be an infinite set of distinct concepts satisfying this witness. 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 v_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 $\ell(p)$.

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.$\leftarrow\bullet$

$\leftarrow\bullet$ Jose: I see that this deviates from a cost metric for programs, but eliminates ties, which is a good thing!

We will in the next three sections use the $\prec$ ordering of programs to define three distinct notions of i) complexity of example sets and ii) teaching size of concepts. We will also discuss, and prove results about, the effectiveness of teaching under these scenarios. We first use Kolmogorov complexity to define K-teaching size TS_K, which corresponds to using $\prec$ just as in the preference-based teaching dimension. We then use Levin's complexity Kt, related to Levin Search, to define Kt-teaching size TS_{Kt}. Finally, we use time-bounded Kolmogorov complexity K^f to define K^f-teaching size TS_{K^f}.

3 Teaching size according to program length preference: TS_ℓ

The ordering $\prec$ can be used to define the first program for a concept according to ℓ and $\prec$ as follows:

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

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 late. This is done by also defining the first program 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:

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

i.e., the teaching size for a concept c using preference ℓ is the size of the smallest witness set S, under Elias encoding, such that the first program in the order of ℓ and $\prec$ satisfying S is in the equivalence class of c.

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

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

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.

However, we are dealing with universal languages and in this case finding the shortest program, i.e., computing the Kolmogorov complexity of a concept, is undecidable, so both learner and teacher would in general also be undecidable. We need another approach.

4 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 [5, 6]. We define the order given by Levin's search for an example set $S = \{< i_1, o_1 >, ..., < i_k, o_k >\}$ as follows:

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

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:

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

$\Phi_{\ell\tau}(S)$ 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:

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

i.e. the size of the smallest witness set S, using Elias 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 $S = \{< i_1, o_1 >, ..., < i_k, o_k >\}$ – note in this case we do not allow a pair $< i, \perp >$ – we want to find $\Phi_{\ell\tau}(S)$, the Levin-simplest program satisfying S.

The learner is computable: given a witness run the learn can perform the original dovetail procedure of Levin's universal search which is 2-dimensional on an increasing budget: over programs of increasing length ℓ following $\prec$ and over increasing runtimes τ, and in this way 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-Concept Book of program/witness pairs (p, w) with p the Levin-simplest program satisfying w, and w the smallest witness for p. A brute-force algorithm to fill the Levin-Concept Book, initially empty, is as follows: try all witnesses w with no contradictory pairs (i.e., no two pairs with distinct outputs for the same input) and no $< i, \perp >$ pairs, in order of increasing Elias encoding size (lexicographically for those of the same size) and run the learner algorithm (i.e., Levin search) on w and then test if the program p found by the learner on this w (using the strategy described above for the learner) is already in the Levin-Concept Book, and if it is not then add the pair (p, w) to the Levin-Concept Book. In that way, the Levin-Concept 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-Concept 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 Kt-teaching size of the concept is in any case well-defined, as the smallest witness. This observation will hold also when we later consider the time-bounded Kolmogorov complexity K^f.

We have the following positive result.

Claim 4.1. *For any universal language L and finite example set S, there will exist* *in the Levin-Concept Book some* (p, w) *with* $p = \Phi_{\ell\tau}(S)$*. If* $p \in Class(c)$ *then* $TS_{\ell\tau}(c) \leq \delta(w)$*.*

Proof. When the teacher tries S then the Levin-simplest program p satisfying S will be found and either p has already been found paired with a smaller witness w or (p, S) will be inserted in the Levin-Concept Book. With $p \in Class(c)$ we have $TS_{Kt}(c) = \delta(w)$ unless there is a program p' equivalent to p that is Levin-simplest for some example set smaller than w. $\square$

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 4.2. *For any Universal Turing Machine (universal language) U there exists another UTM U' s.t. no id-program will be in the Levin-Concept 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. 1) Make a computable set $\{p_i :$ binary string $i\}$ of programs such that $p_i(x) = x$ except for $p_i(i) \neq i$, and let this be fast. S2) 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. $\square$

Thus, even for quite simple concepts we cannot avoid that there will be some contrived universal languages that 'fool' the Kt-teacher so that a witness set for this concept cannot be found. To avoid this we need an alternative definition of teaching size.

5 Teaching size TS_{K^f} based on time-bounded Kolmogorov complexity

We will now use a time complexity function $f : \mathbb{N} \rightarrow \mathbb{N}$, the idea being that if the learner is given an example set $S = \{< i_1, o_1 >, ..., < i_k, o_k >\}$ with $bits(S) = \sum_{1 \leq j \leq k} |i_j| + |o_j|$ when a program has run on an input for $f(bits(S))$ ←● steps without halting then the learner simply aborts the program. Note that we now allow $< i, \perp > \in S$. 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 $< i, \perp >$.

←● JA: Note this looks strange, as we allow more time on an input i if this i is part of a large S. It is done to ensure there is always some program satisfying S, see below. I don't think it hurts our results, but I am not sure...

We say that a program p is f-compatible with example set S, denoted $p \models_f S$, if for all pairs $< i, o > \in S$ the program p on input i will within $f(bits(S))$ 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.

Clearly, using larger time complexity functions we will be able to correctly discriminate between more programs. For a time complexity function f, the time-bounded Kolmogorov complexity of an example set S is defined as

$$K^f(S) = \min_{p \models_f S} \{\ell(p)\}$$

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

$$TS_{K^f}(c) = \min_S \{\delta(S) : argmin_{\prec}\{K^f(S)\} \in Class(c)\}$$

if such S exists and otherwise undefined; i.e. the K^f-teaching size for a concept c is the size of the smallest witness set S (using Elias 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 K_f-teaching size will not be defined.

K^f-teaching size could alternatively be defined using $\prec$ as preference-based order directly:

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

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.

To ensure that for every example set S there is some f-compatible program we require that all our time complexity functions satisfy what we call the hard-coded linear bound $f(bits(S)) \geq c_L * bits(S)$. ←● The constant c_L, dependent on the language L, is chosen so that $c_L * bits(S)$ is larger than the time taken, on any input i_j in $S_{out} = \{< i_1, o_1 >, ..., < i_k, o_k >\}$, for which $o_j \neq \perp$, by some program hard-coding these i/o pairs of S_{out}. For example the program consisting of a large if-then-else-if giving the correct output on each of these i/o-pairs, and otherwise entering an infinite loop, which will thereby make this hard-coded program f-compatible with S.

←● JA: NB This assumes that on an input i_j in S the hard-coded program for S does not take more time than linear in bits(S). Are there languages where this is not the case?

When the learner is given an example set S we want to find $argmin_{\prec}\{K^f(S)\}$, the smallest program f-compatible with S. 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. There is no need for applying Levin search, since the time limit will ensure we never run into an infinite loop. We ensure that S does not contain any contradictory pair (two distinct behaviors on the same input), and that f obeys the hard-coded linear bound. This ensures that there will always be a program f-compatible with S.

Let us now turn to the teacher. The teacher is computable, by filling what we call an f-Concept Book 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-Concept Book, initially empty, is as follows: try all example sets S with no contradictory pairs but allowing $< i, \perp >$ pairs, in order of increasing Elias encoding size (lexicographically for those of the same size) 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-Concept Book, and if it is not then add the pair (p, S) to the f-Concept Book. The hard-coded linear bound is crucial so that for any S the learner will find some f-compatible program.

←●

←● JA: I removed the result showing that we can teach any finite behavior, i.e. a similar result as for Levin search. Its commented out in the tex file.

We show that we can, for any concept, partial or total, teach this concept by carefully choosing the right time complexity function.

Theorem 1. *For any universal language L, and for any L-concept c, there is an $f()$ so that the f-Concept Book will contain some (p_c, w_c) with $p_c \in Class(c)$ and $TS_{K^f}(c) = \delta(w_c)$.*

Proof. Given the language L and an L-concept c we take p the smallest program equivalent to c and define a time complexity function $f()$, ensuring it obeys the hard-coded linear bound, to show that the teacher will find a pair (p, w) for insertion in the f-Concept Book. Note this will suffice to prove the theorem; if $TS_{K^f}(c) < \delta(w)$ there is some smaller w_c with $TS_{K^f}(c) = \delta(w_c)$ for which p_c, the shortest program f-compatible with w_c, is equivalent to p (it may be that $p = p_c$) and when the teacher reached w_c the pair (p_c, w_c) would be inserted in the f-Concept Book. We know the teacher reaches w_c in finite time since it tries example sets in increasing size order and f obeys the hard-coded linear bound so that for any example set there is some f-compatible program.

For partial concepts we allow the witness w to contain i/o-pairs consisting of $< i, \perp >$ 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 $< i, o >$ and in the second case use $< i, \perp >$. We thus have $w = w_\perp \cup w_{out}$ consisting of q pairs $w_\perp = \{< i_1, \perp >, ..., < i_q, \perp >\}$ with q programs $\{p_1, ..., p_q\}$ for which these are conflicting pairs, thus $p_j(i_j) \neq \perp$ for $1 \leq j \leq q$, in addition to k pairs $w_{out} = \{< i_{q+1}, o_{q+1} >, ..., < i_{q+k}, o_{q+k} >\}$ where the output is specified.

We now chose any non-decreasing time complexity function f that obeys both the hard-coded linear bound and 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 τ is the number of steps executed by a program on a specified input. This will ensure that, for each $< i, \perp > \in w_\perp$ being a conflicting pair for some p', we have f big enough to allow $p'(i)$ to run to completion, and for $< i, o > \in w_{out}$ we have f big enough to allow $p(i)$ to run to completion, since $|i| \geq bits(w)$ and f non-decreasing.

As the teacher starts filling the f-Concept 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 $< i, o > \in w$, if $< i, o > \in w_{out}$ then it will be discovered that p' does not output o within time $f(bits(w))$ (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 $< i, \perp > \in w_\perp$, 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 $< i, o > \in w$ it will within time $f(bits(w))$ output the correct thing on input i, while for a pair $< i, \perp > \in w$ it will within time $f(bits(w))$ 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-Concept Book it will be inserted. Since the teacher algorithm goes through witnesses by increasing size order, the f-Concept Book will contain the witness with smallest size that leads the learner to output p, and this is not necessarily equal to the w we have specified in the proof. $\square$

Since the L-concepts 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 doesn't halt. Therefore, the result we have just shown goes beyond teaching the set of computable functions, i.e. the total functions computed by some program, which by definition must always halt.

In case an L-concept c 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 to strengthen the result to teach the whole class of concepts computable within the same time bound. Fix any non-decreasing $t : \mathbb{N} \rightarrow \mathbb{N}$ and define the class C_L^t of total L-concepts having implementations with runtime $t(n)$:

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

Theorem 2. *For any t and any L, using time complexity function defined by $f(n) = \max\{t(n), c_L * n\}$ for all n, we can teach the concept class C_L^t. For any $c \in C_L^t$ the f-Concept Book will contain some (p, w) with $p \in Class(c)$ and $TS_{K^f}(c) = \delta(w)$.*

Proof. Consider any concept $c \in C_L^t$ and its equivalent program p_c whose runtime is upper-bounded by $t(n)$. We use an analogous proof as in Theorem 1, to find a witness $w = \{< i_1, o_1 >, ..., < i_k, o_k >\}$ 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 $< i, \perp >$, 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 it obeys both the hard-coded linear bound and obeys that for each $1 \leq j \leq k$ we have $f(|i_j|) \geq t(|i_j|) \geq \tau(p_c, i_j)$, and since $|i_j| \geq bits(w)$ and t non-decreasing this 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 are never higher than $f(n) = \max\{t(n), c_L * n\}$ for all n, where the latter is the hard-coded linear bound.

The remainder of the proof proceeds in a manner analogous to the proof of Theorem 1. □

6 Sampling prior

Note that with learning prior K^f the algorithm for filling the f-Concept Book will for any witness w 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 w with no contradictory pairs, and the hard-coded linear bound, so there is always some program compatible with all pairs. However, the teacher will never finish filling the f-Concept Book, as there is an infinity of distinct concepts corresponding to programs, and thus no upper bound on their teaching size.

To work in a finite setting and formalize a sampling prior we set a size-limit on the maximum witness size at which we halt computation of the f-Concept Book. We thus compute the initial part of the infinite concept book consisting of concepts with smallest teaching size. We can then order concepts in the f-Concept Book by increasing teaching size and define a *sampling prior* respecting this ordering. Thus, a concept whose witness set has small total size will be more likely to be chosen by the teacher. Concepts whose teaching size is above the size-limit were not added to the f-Concept Book and will have probability zero but any concept with teaching size within the size-limit will have positive sampling probability.

Note that in the resulting f-Concept Book the majority of concepts will usually have teaching size close to the size-limit, as there are more large witnesses. However, the distribution can be always chosen to decay at a rate fast enough, or slow enough, so that we get any desired *expected* teaching size.

We can also play with various definitions of ”interesting” concepts and give higher prior to these.

7 Experimental validation

We describe an experimental validation of our method for teaching a universal language using a learning prior. We work with the universal language P3, a simple language for string manipulation. As learning prior we take the preference order $\prec$ built on program size and time-bounded Kolmogorov complexity. We will thus be computing the K^f-teaching size TS_{K^f} of P3-concepts, i.e. concepts computable by P3 programs. We implement the teacher algorithm described in Section 5.

Thus, the teacher considers witnesses by increasing size and for each witness considers programs in $\prec$ order. For each input/output pair (i, o) of the witness w 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, w) pair is inserted into the f-Concept Book unless p has already been matched with a smaller witness. When all witnesses up to a fixed size bound have been tried we halt the procedure, thus computing the initial part of the infinite concept book consisting of concepts with smallest teaching size.

To compare the two notions of teaching size and teaching dimension we run two experiments. In the first experiment we compute the K^f-teaching size TS_{K^f} as described above, with the teacher considering witnesses consisting of several i/o pairs. In the second experiment the teacher considers only witnesses consisting of a single i/o pair, generating these by increasing size, thus filling the TD1-concept 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 2 weeks of computing time for filling the concept book. $\leftarrow\!\bullet$ Table 1 shows that the approach based on teaching size (TS) is the more efficient since it has: a higher (constant) time complexity function f, smaller size in bits of the max witness considered, more example sets tested, more distinct programs in the resulting context book. We start by describing the language P3, then more details of the two experiments and a closer comparison of the results.

$\leftarrow\!\bullet$ JA: Say something about the computer used?

Table 1: Two experiments: Teaching Size and Teaching Dimension 1

Name	**Days**	**Time f**	**Bit-size**	**Example sets**	**Programs**
TS	14	2000	7	17.252	5.062
TD1	14	200	9	9.217	3.227

7.1 The P3 language

P" is a primitive programming language introduced by Corrado Böhm [1] in 1964, and was the first GOTO-less imperative language proved Turing-complete, i.e. universal. One of its better known variants has 8 instructions total (`https://en.wikipedia.org/wiki/Brainfuck`). 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 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, as long as the content of the pointer cell is not '.'.

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 to define the $\prec$ length-lexicographic ordering of P3 programs '<>+-[]o'. 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 program first moves the pointer one position to the left: "<", and then returns the pointer to the "." to 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".

The example set (('0100', '1000')('00010', '00100')), i.e. with two i/o-pairs, will find this program, meaning that no shorter program has this particular i/o-behavior. 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 $\prec$ the left shift P3 program has teaching dimension two.

7.2 Teaching Size

In this section we present the results of running the teacher algorithm to compute the concept book when witnesses consist of several pairs, thus computing the TS-concept book of P3 programs ordered by K^f-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 it is met only by non-halting programs. ←● Table 2 gives some (p, w) pairs, of P3 programs and example sets, that the teacher will find.

←● JA: Is this a true statement?

Table 2: (p, w) pairs, of P3 programs and example sets, found by the teacher.

Example set	Program	Description
('0', '0'), ('10', '10')	[o>]	identity
('010000', '000010') ('1000', '0001')	[>]+[<o]	reverse
('011', '11') ('10001', '11')	[-[+o+]>]	filter 0
('011', '0') ('10001', '000')	[+[-o-]>]	filter 1
('01', '10') ('0011', '1100')	[+[o>+]+o]	swap 1 and 0
('01', '11') ('0011', '1111')	[[+]-o>]	convert 0 to 1
('01', '00') ('0011', '0000')	[[+]+o>]	convert 1 to 0
('0100', '00') ('001', ")('00', '00')	[+>]<[-o<]	remove before last 1
('0100', '1000')('00010', '00100')	>[o>]<[<]>o	left shift

When running the teacher on all possible example sets of increasing size in order to fill a concept book we choose to limit the size of example sets to at most 7 bits. Let us see a procedure to do this and give an upper bound on the number of example sets. Assume we have a binary string of size k and consider how many ways this string can be broken up to form an example set. If $k = 0$, both input and output have size zero: $(|0|, |0|)$. If $k = 1$, there are 2 possible ways to break this into an example set, of i/o sizes: $(|0|, |1|)$ and $(|1|, |0|)$, giving the $2 * 2^1$ example sets $(, 0), (0,), (, 1), (1,)$. If $k = 2$, the possible combinations of (input,output) sizes in the example set is: $(|1|, |1|), (|2|, |0|), (|0|, |2|)$, $((|1|, |0|), (|0|, |1|))$, $((|0|, |1|), (|1|, |0|))$, $((|1|, |0|), (|1|, |0|))$, $((|0|, |1|), (|0, ||1|))$, for a total of 7×2^2. However, note that we are overcounting, e.g. the last example set either has two repeated pairs (when the two bits are equal) or else a contradictory pair (with different outputs for one input), so this gives only an upper bound. The general expression is $f(k) \times 2^k$, where the function f is the sum of the number of submultisets of the multisets with n elements [1]. We have $f(3) = 18, f(4) = 50, f(5) = 118, f(6) = 301, f(7) = 684$, for a total of 111.569 when summing up to size 7. From this set we remove 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 we are left with 17.252 example sets.

For each of the 17.252 example sets, a P3 program was found, with some programs identified by many example sets. The final TS-concept book contains 5.062 distinct programs and their witnesses, i.e. the smallest example set that identifies them. In Table 3, we show three examples of witness sets and its corresponding P3 program in the TS-concept book.

Table 3: Some programs and witnesses belonging to the TS-concept book.

Witness	**Program**
((, 1), (0, 01))	[o>]
((, 0), (0,), (10,))	-[-o]
((, 1), (0, 01), (10, 1))	[o>>]<-o
((, 1), (0, 11), (1, 10))	[>+<+]-o>o

Figure 7.2 shows that the majority of the 5.062 programs in the TS-concept book have witnesses with more than one i/o-pair.

7.3 Teaching dimension 1

In this section we present the results of running the teacher algorithm to compute the concept book when witnesses consist of a single i/o-pair, thus computing the TD1-concept book of P3 programs that under the $\prec$ ordering have teaching dimension 1. We choose a constant time limit function $f(n) = 200$ for all n, since the hard-coded P3-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 bits, since this is what we could manage within the total computation

[1]See OEIS: `https://oeis.org/A074141`.

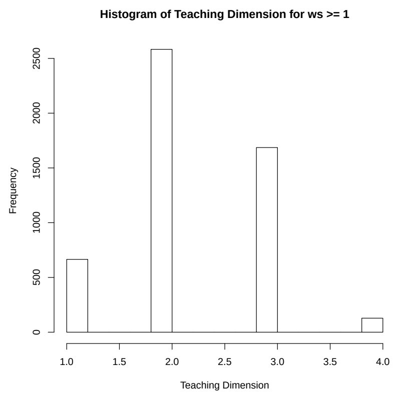

Figure 1: Number of i/o pairs in witnesses of the TS-concept book

time, 14 days, allowed for the previous experiment. Let us see a procedure to do this and count the number of example sets. Assume we have a 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 TD1-concept book contained 3.227 distinct programs and their witnesses. Table 4 shows some programs and witnesses in the TD1-concept book.

Table 4: Some program and witness pairs in the TD1-concept book.

Input	**Output**	**Program**
110	010	-[o>]
00	110000	>>-[oo<]
	0010101	>>>+[o<+o+]
	11110000	-[oooo-]

Figure 3 gives a correlation matrix comparing the learning steps (total time for teacher or learner to find the program given a witness), size of the input, size of the output←●, length of the program, and the number of example sets identifying a program. This allows to make some observations. First, length of programs is highly correlated with the size of outputs and inversely correlated with size of inputs. When we have short inputs the identified program is usually one that more or less hard-codes the output. Given an output of size k, we will need at

←● Jose: Let's use size for examples and length for programs worst 2 $\times k$ instructions to obtain the desired output: one instruction to get the correct value in the pointer cell ("+" or "-"), and the "o" instruction. The length of the programs seem to follow a normal distribution. The median seems to be 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.

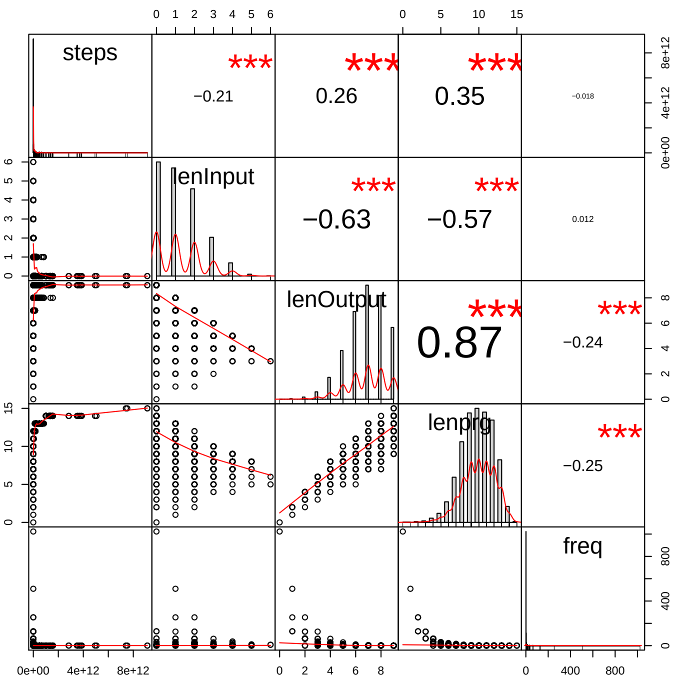

Figure 2: Correlation matrix, for TD1, comparing learning steps, size of the input, size of the output, length of the program, and number of example sets identifying the program.

7.4 Comparing the two approaches: teaching size vs dimension

Let us first look at the size of the programs in the TS-concept book versus the TD1-concept book. In Figure 7.4 we see that the programs in the TS-concept book are generally smaller. The TD1-concept book contains many (long) hard-

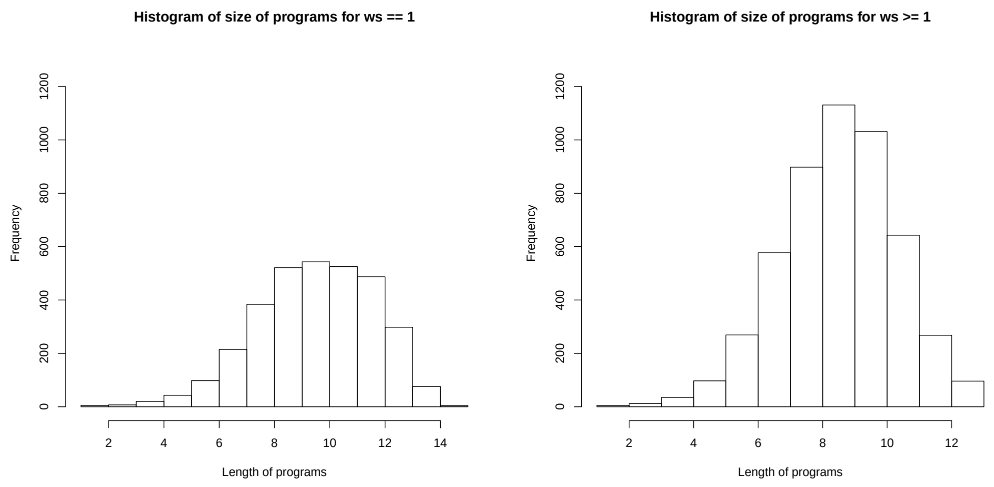

Figure 3: Number of instructions of the programs in the TD1-concept book and the TS-concept book.

coded programs, since hard-coding is more prevalent when having to satisfy only a single i/o-pair.

543 programs were common to both concept books. For each of these programs, we want to compare the size of their witness set in the two concept books. However, we cannot compare size simply by number of bits since we must account for the separation into several i/o-pairs. In order to make a fair comparison possible, we use the cost in bits of encoding each binary string in the witness sets using the Elias delta coding [2], as done previously. Since these are prefix codes their concatenation preserves the separation into the original binary strings.

In Figure 4 we include a scatter plot where each point (or line) represents a group of these 543 programs, sharing the same pair of sizes of witness encodings. The gray scale of points and lines represent the number of programs in the group, with darker denoting a higher number. The x axis represents the Elias coding of the witness set in the TD1-concept book, while the y axis represents the Elias coding of the witness set in the TS-concept book. As expected, no programs are above the diagonal, as the TS-concept book is built to minimize size. The horizontal lines represent programs present in the TS-concept book, but not present in the TD1-concept book. ←● ←● ←● Since we do not know their size in the x axis, or whether these programs have teaching dimension 1 at all, we represent them as horizontal lines, giving only their TS size.

←● JA: I forget, is this what the lines actually represent?

←● Cesar: Yes, we know that their Elias Code will be greater than the diagonal.

←● JA: My question was: are those lines ALL the programs in TS but not in TD1?

For programs on the diagonal the two sizes in Elias coding are identical, but there are some exceptions (17 programs) where the size in the TS-concept book is smaller, see Table 5. For example, the program $+[o >] - o$ which is basically 'increase, output while 0 or 1, decrease and output' has the TD1 witness (0000,10001) and the TS witness ((,10),(1,1)). The difference in bit size is 9 to 4 and in Elias encoding 19 to 14.

Table 5: Examples of programs in both concept books where the witness size using Elias coding is smaller in TS than in TD1.

TD1	Cost	**TS**	Cost	**Program**
(0000,10001)	19	((, 01), (1, 1))	14	+[o>]-o
(11,0001110)	19	((, 1110), (0, 0))	18	-[ooo>]+o
(1001,00010)	19	((, 10), (0, 0)	14	-[o>]+o

8 Discussion

References

[1] Corrado Böhm. On a family of turing machines and the related programming language. *ICC Bull*, 3(3):187–194, 1964.

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

[3] Ziyuan Gao, Christoph Ries, Hans U Simon, and Sandra Zilles. Preference-based teaching. In *Conf. on Learning Theory*, pages 971–997, 2016.

[4] Jose Hernandez-Orallo and Jan Arne Telle. Finite biased teaching with infinite concept classes. *arXiv preprint arXiv:1804.07121*, 2018.

[5] Leonid A. Levin. Universal Search Problems. *Problems Inform. Transmission*, 9:265–266, 1973.

[6] Ming Li and Paul Vitányi. *An Introduction to Kolmogorov Complexity and its Applications*. 3rd Ed. Springer, 2008.

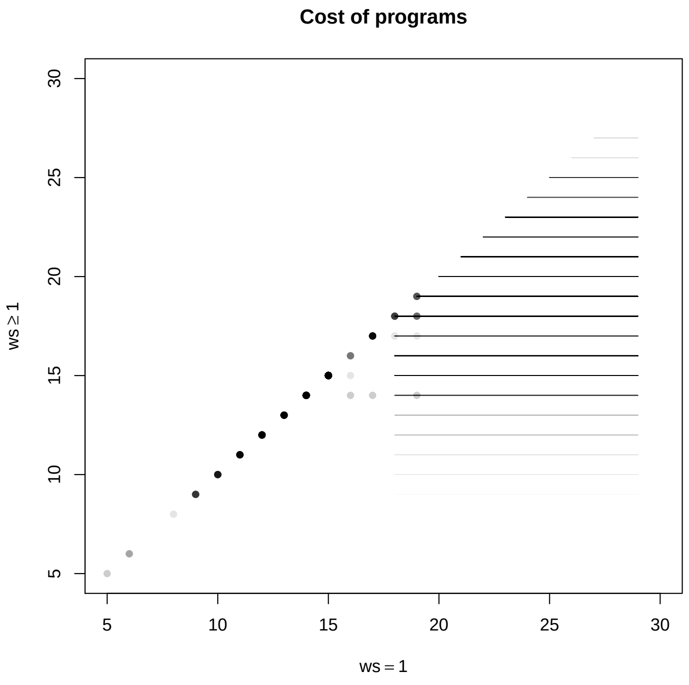

Figure 4: Comparing Elias coding size of witnesses in TD1 (x-axis) and TS (y-axis)