Machine Learning (2023) 112:4049–4080
https://doi.org/10.1007/s10994-023-06347-4

Heuristic search of optimal machine teaching curricula

Manuel Garcia-Piqueras[1] · Jose Hernández-Orallo[2]

Received: 1 August 2022 / Revised: 13 April 2023 / Accepted: 8 May 2023 /
Published online: 12 June 2023

Abstract

In curriculum learning the order of concepts is determined by the teacher but not the examples for each concept, while in machine teaching it is the examples that are chosen by the teacher to minimise the learning effort, though the concepts are taught in isolation. Curriculum teaching is the natural combination of both, where both concept order and the set of examples can be chosen to minimise the size of the whole teaching session. Yet, this simultaneous minimisation of teaching sets and concept order is computationally challenging, facing issues such as the "interposition" phenomenon: previous knowledge may be counter-productive. We build on a machine-teaching framework based on simplicity priors that can achieve short teaching sizes for large classes of languages. Given a set of concepts, we identify an inequality relating the sizes of example sets and concept descriptions. This leverages the definition of admissible heuristics for A* search to spot the optimal curricula by avoiding interposition, being able to find the shortest teaching sessions in a more efficient way than an exhaustive search and with the guarantees we do not have with a greedy algorithm. We illustrate these theoretical findings through case studies in a drawing domain, polygonal strokes on a grid described by a simple language implementing compositionality and recursion.

Keywords Curriculum learning · A* · Optimisation · Interposition

Editors: Alireza Tamaddoni-Nezhad, Alan Bundy, Luc De Raedt, Artur d'Avila Garcez, Sebastijan Dumanić, Cèsar Ferri, Pascal Hitzler, Nikos Katzouris, Denis Mareschal, Stephen Muggleton, Ute Schmid.

Manuel Garcia-Piqueras
manuel.gpiqueras@uclm.es

Jose Hernández-Orallo
jorallo@upv.es

[1] Department Mathematics, Universidad de Castilla-La Mancha, C. Altagracia, 50, 13001 Ciudad Real, Spain

[2] VRAIN, Universitat Politècnica de València, C/de Vera s/n, 46022 Valencia, Spain

1 Introduction

Humans are able to learn a new concept from very little information. However, a large majority of machine learning algorithms needs a substantial amount of data to perform similarly. In particular, successful models such as deep learning are among the most data-hungry (Marcus, 2018). Additionally, humans are able to identify more complex object categories than machines by composing simpler objects, but the best deep learning techniques usually struggle in compositional settings (Shrestha & Mahmood, 2019). How is it possible that humans perform such a rich learning from so scarce information? (Lake et al., 2015)

In essence, education in humans is organised through curricula, starting with simple concepts and gradually increasing the complexity. Bengio et al. (2009) initiated the formalisation of curriculum learning (CL) in the context of machine learning. It inspired new applications in machine learning to an extensive variety of tasks (Tang et al., 2018; Wang et al., 2018, 2019). The empirical results showed advantages of CL over random training. Nevertheless, CL is still limited by two issues: (1) the need to find a way of ordering the examples according to their difficulty and (2) the correct order of the concepts involved (Soviany et al., 2022).

Machine teaching (MT) is the area of AI that looks for optimal examples that a teacher should utilise to make a student identify a concept (Zhu et al., 2018). It is of extreme importance in applications such as digital assistants, where we would like to teach them new concepts or procedures with limited data, in the same way humans teach or communicate with other humans (Degen et al., 2020).

MT is often understood as an inverse problem to machine learning (Zhu, 2015). For one single concept machine teaching works as follows: the teacher generates a training—or witness—set of examples, either positive or negative, from which the learner identifies a concept. For instance, consider a teacher who wants a student to acquire the concept "red ball"; the teacher knows the student's learning algorithm and may provide it with the following witness set: a red tennis ball picture labelled as positive and a blue one labelled as negative.

For humans and many other animals it is assumed that once a concept has been captured it is possible to reuse it to learn another one. Previous work (Zhou & Bilmes, 2018) has tried to optimise the teaching of one concept, e.g., through a partial minimax approach. However, not only do we want to consider the sequence of training sets for a concept, but also the best order to teach a given set of concepts. MT for several concepts was restricted to sequences where every acquired concept becomes background knowledge (Clayton & Abbass, 2019; Wang et al., 2021). Pentina et al. (2015), who had studied CL from an experimental approach, proposed that future work should identify a valid theoretical framework that would allow for a more generic distribution of tasks and the realisation of the advantages of forgetting or using independent sessions. This is in line with neurological studies such as Richards and Frankland (2017) and recent animal cognition research (Dong et al., 2016; Epp et al., 2016; Shuai et al., 2010), showing "evidence that forgetting is necessary for flexible behavior in dynamic environments".

The first full theoretical framework of CL in MT was introduced in Garcia-Piqueras and Hernández-Orallo (2021). Using simplicity priors it identifies the optimal tree-distribution of concepts by means of the 𝕀-search algorithm (Garcia-Piqueras & Hernández-Orallo, 2021). The framework defines an instructional curriculum as the set of alternative partial sequences, such as the upper and lower branch of Fig. 1.

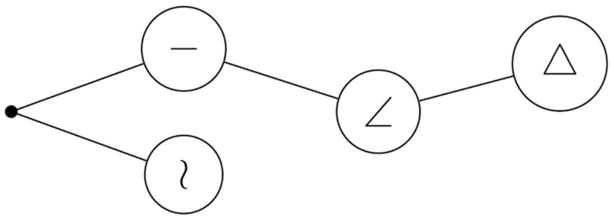

Fig. 1 Instructional curriculum for a set of concepts, where concept − is taught before ∠ and these two before △, in one session, and concept ≀ is taught in an independent session or lesson

The order between the branches is irrelevant, but the order of the concepts in each branch is crucial. The MT framework implementing CL proposed in Garcia-Piqueras and Hernández-Orallo (2021) not only meets the specifications in Pentina et al. (2015), but is also consistent with Richards and Frankland (2017) by handling a new phenomenon called interposition: previous knowledge is not always useful. This issue increases the difficulty of finding optimal curricula.

Under general conditions, $\mathbb{I}$-search is able to overcome interposition, but it is computationally intractable in general. The computational cost is one of the reasons why CL is not sufficiently used in AI (Forestier et al., 2022), despite existing solutions with search algorithms like A^* (Pearl, 1984). Heuristic estimates enhance those procedures by making less node expansion in the graph search, which drastically reduces computational costs (Rios & Chaimowicz, 2010). To our knowledge, there are no such estimators for CL.

Here we introduce new theoretical results, such as Inequality (2): a relation between the sizes of the examples and concept descriptions with and without background knowledge. Such inequality is key to define a new family of heuristics to effectively identify minimal curricula. The heuristics are elegantly defined using a "ratio of similarity" between the sizes of the sets of examples with and without previous knowledge. Such ratio is the quotient of the lengths of the descriptions of a concept with and without background knowledge.

Theoretical results are illustrated on a drawing domain, where curricula can exploit that some drawings built on substructures previously learnt (e.g., a flag is depicted as a rectangle and a straight pole). Experiments show the effect of interposition in CL and how it is overcome through our novel approach. This contribution, based on compositional simplicity priors and exemplified using a drawing domain, follows the direction of other important efforts in compositional AI (Lake et al., 2015; Wu et al., 2016; Tenenbaum et al., 2000; Wong et al., 2021).

2 The machine teaching framework

In this section we give more details about the interaction between the *teacher* and the *learner* as distinct entities. The teacher-learner protocol (Telle et al., 2019) is based on the following statements. Let Σ^* be the set of all possible strings formed from combinations of symbols of an alphabet Σ. We label such strings as *positive* or *negative*. An example is an input–output pair where the input is a string of Σ^* and the output is + or − (outputs might be strings in the most general case of the framework).

Definition 1 The example (or instance) space is defined as the infinite set

$$X = \left\{ \langle \mathtt{i}, \mathtt{o} \rangle : \langle \mathtt{i}, \mathtt{o} \rangle \in \Sigma^* \times \{+, -\} \right\}$$

There is a total order $\lessdot$ in X and there is a metric δ that gives the size of any set of examples.

Definition 2 We define the infinite concept class $C = 2^X$ consisting of concepts that are a subset of X.

The objective is that for any concept $c \in C$ the teacher must find a *small* witness set of examples from which the learner is able to uniquely identify the concept. The learner tries to describe concepts through a language L.

Definition 3 A program p in language L satisfies the example $\langle \mathtt{i}, \mathtt{o} \rangle$, denoted by $p(\mathtt{i}) = \mathtt{o}$, when p outputs $\mathtt{o}$ on input $\mathtt{i}$[1] . We say that a program p is compatible with the set of examples $S \subset X$, if p satisfies every example of S and we denote $p \vDash S$.

Two programs are equivalent if they compute the same function mapping strings of Σ^* to $\{+, -\}$. There is a total order $\prec$ defined over the programs in L. For any program p in L there is a metric ℓ that calculates its length.

We say that c is an L-concept if it is a total or partial function $c : \Sigma^* \to \{+, -\}$ computed by at least a program in L. Let C_L be the set of concepts that are described by L. Given $c \in C_L$, we denote $[c]_L$ as the equivalence class of programs in L that compute the function defined by c.

Definition 4 We define the first program, in order $\prec$, returned by the learner Φ for the example set S as

$$\Phi_\ell(S) = \arg\min_p{}^{\prec} \{\ell(p) : p \vDash S\}$$

We say that w is a *witness set* of concept $c \in C_L$ for learner Φ, if w is a finite example set such that $p = \Phi_\ell(w)$ and $p \in [c]_L$.

The teacher Ω has a concept $c \in C_L$ in mind and knows how the learner works. With this information, the teacher provides the learner with the witness set w.

Definition 5 We define the *simplest* witness set that allows the learner to identify a concept c as

$$\Omega_\ell(c) = \arg\min_S{}^{\lessdot} \left\{ \delta(S) : \Phi_\ell(S) \in [c]_L \right\}$$

We define the teaching size of a concept c as $TS_\ell(c) = \delta(\Omega_\ell(c))$.

We exemplify our discourse with a drawing domain: polygonal strokes on a grid. The details about the definition of a language L and a example space X are given in Section A. In short, we deal with an example space generated by commands North, South, East and

[1] The general framework also considers when the program does not halt by means of complexity functions.

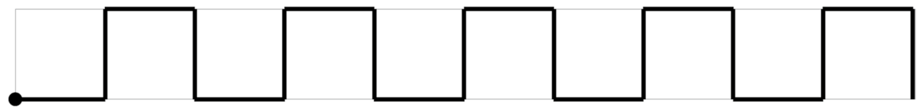

Fig. 2 Polygonal on a grid

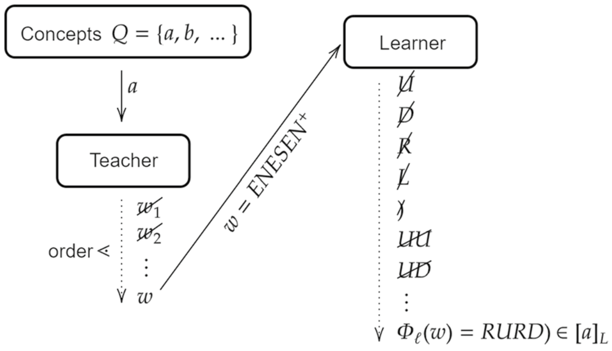

Fig. 3 The teacher-learner protocol for a given concept a

West, for the four possible axis-parallel directions in a plane. In Fig. 2, starting at the black dot, the frieze is described by ENESENES … Examples are labelled positive or negative, e.g., EN^+ or E^-.

The learner is able to capture concepts using a language L with the following instructions: Up, Down, Right and Left,) and @. The symbol) refers to a non-deterministic choice between going to the beginning of the program or continue with the next instruction. For instance, the concept a defined by the frieze in Fig. 2, could be expressed in L as RURD). The instruction @ stands for library calls to implement background knowledge; for instance, RU@ is equivalent to RURD) when the library points to subroutine RD).

For example, the teacher receives or thinks about such concept a. The teacher selects the witness set, $w = \{\text{ENESEN}^+\}$ and provides the learner with it. At that point, the learner outputs the first program that satisfies w, i.e., $\Phi_\ell(w) = \text{RURD)} \in [a]_L$ (see Fig. 3). We usually drop the index ℓ if it is clear from the context. The teaching size of concept a is the size of the witness set, i.e., $TS(a) = \delta(\{\text{ENESEN}^+\}) = 21$ (3 bits per symbol as it is stated in Section A).

As we have seen, the teacher-learner protocol makes it possible to define the *teaching size* of a concept. The MT framework was adapted to implement background knowledge through the notion of *conditional teaching size* (Garcia-Piqueras & Hernández-Orallo, 2021). We discuss this approach in the following section, along with a phenomenon called *interposition*.

3 Conditional teaching size and interposition

Let a library be a possibly empty ordered set of programs. We denote $B = \langle p_1, \ldots, p_k \rangle$, where each p_i identifies concept c_i and $|B|$ denotes the number of primitives k. We use $|B| = 0$ to indicate that B is empty. A program p, identified by the learner, might be included in a library B as a primitive for later use.

Table 1 Conditional teaching sizes for concepts of Example 1

TS (bits)	Teacher-Learner identification
$TS(a) = 21$	$\Phi(\{\mathsf{ENESEN}^+\}) = \mathsf{RURD)}$
$TS(b) = 24$	$\Phi(\{\mathsf{EENESEE}^+\}) = \mathsf{RRURD)}$
$TS(c) = 24$	$\Phi(\{\mathsf{ENESEEE}^+\}) = \mathsf{R)URD)}$
$TS(a\|b) = 21$	$\Phi(\{\mathsf{ENESEN}^+\}\|B) = \mathsf{RURD)}$
$TS(b\|a) = 12$	$\Phi(\{\mathsf{EEN}^+\}\|B) = \mathsf{R@}$
$TS(c\|a) = 24$	$\Phi(\{\mathsf{ENESEEE}^+\}\|B) = \mathsf{R)URD)}$
$TS(a\|c) = 30$	$\Phi(\{\mathsf{ENESEN}^+, \mathsf{EE}^-\}\|B) = \mathsf{RURD)}$
$TS(b\|c) = 36$	$\Phi(\{\mathsf{EENESEE}^+, \mathsf{EEE}^-\}\|B) = \mathsf{RRURD)}$
$TS(c\|b) = 24$	$\Phi(\{\mathsf{EENESEN}^+\}\|B) = \mathsf{R)URD)}$
$TS(a\|b, c) = 30$	$\Phi(\{\mathsf{ENESEN}^+, \mathsf{EE}^-\}\|B) = \mathsf{RURD)}$
$TS(b\|a, c) = 21$	$\Phi(\{\mathsf{EEN}^+, \mathsf{EN}^-\}\|B) = \mathsf{R@0}$
$TS(c\|a, b) = 24$	$\Phi(\{\mathsf{ENESEEE}^+\}\|B) = \mathsf{R)URD)}$
$TS(a\|c, b) = 30$	$\Phi(\{\mathsf{ENESEN}^+, \mathsf{EE}^-\}\|B) = \mathsf{RURD)}$
$TS(b\|c, a) = 24$	$\Phi(\{\mathsf{EEN}^+, \mathsf{EEE}^-\}\|B) = \mathsf{R@1}$
$TS(c\|b, a) = 24$	$\Phi(\{\mathsf{ENESEEE}^+\}\|B) = \mathsf{R)URD)}$

In order to avoid *old references* when the library is expanded, we replace every instruction @ of a program identified by the learner by the corresponding primitive. For example, if $B = \langle \mathsf{D)} \rangle$ and the learner identifies RU@, then the library is extended as $B = \langle \mathsf{D)}, \mathsf{RUD)} \rangle$.

We define the conditional teaching size of concept c, using library B (background knowledge), as the size in bits of the first witness set w such that $\Phi_\ell(w|B) = p \in [c]_L$. Let us see an example.

Example 1 Let us consider $Q = \{a, b, c\}$, where programs $\mathsf{RURD)} \in [a]_L$, $\mathsf{RRURD)} \in [b]_L$ and $\mathsf{R)URD)} \in [c]_L$ (polygonal chains as shown in the 2nd, 3rd and 4th rows of Fig. 6, respectively in Section A). These programs are placed first in their equivalent classes regarding the total order $\prec$.

Graphical instances of a and b are, for example, the polygonal chains of the 2nd and 3rd rows of Fig. 6, respectively. We get $TS(b|a) = 12$ as the teaching size, in bits, of concept b using a as prior knowledge, i.e., the learner employs library $B = \langle \mathsf{RURD)} \rangle$ as background knowledge. In our particular case, the learner outputs $\Phi(\{\mathsf{RRN}^+\}|B) = \mathsf{R@}$ when using library $B = \langle \mathsf{RURD)} \rangle$.

In such case, there is a reduction of the teaching size of concept b using a as prior knowledge: $TS(b|a) = 12 < 24 = TS(b)$ (see Table 1). However, it may happen that background knowledge increases the teaching size of an object. This phenomenon is what we call *interposition*: some prior knowledge causes interposition to a given concept.

For instance, let us consider $\mathsf{R)URD)} \in [c]_L$, with graphical instances such as the chain of the fourth row of Fig. 6. Using library $B = \langle \mathsf{R)URD)} \rangle$, we get that $\Phi(\{\mathsf{ENESEN}^+, \mathsf{EE}^-\}|B) = \mathsf{RURD)}$ (see Table 1), so that:

$$TS(a|c) = 30 > 21 = TS(a)$$

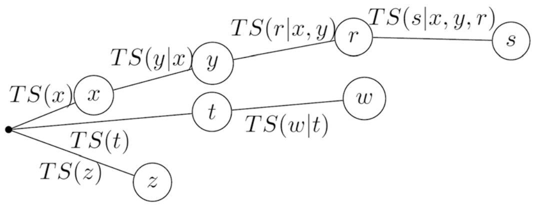

Fig. 4 Curriculum $\{x \to y \to r \to s, t \to w, z\}$ for a set of concepts $\{x, y, r, s, t, w, z\}$

In other words, concept c causes interposition to concept a. This phenomenon increases the difficulty of finding the curriculum with minimum overall teaching size for a given set of concepts. We must take this issue into account in the following sections.

4 An optimisation problem: minimal curricula

In this section, we will deal with the following problem: if we want to teach a given set of concepts, which curriculum minimises the overall teaching size?

In our approach, given a set of concepts, a curriculum is a set of disjoint sequences covering all the concepts. Our view of curricula is more general than just simple sequences. Our choice is motivated by the interposition phenomenon seen in the previous section, since some concepts may increase the teaching size of some other concepts coming after those previous ones. If some branches are disconnected, a curriculum should not specify which branch comes first, since they are considered independent *lessons*. We will show below how the algorithms that find optimal curricula manage this flexibility.

For instance, Fig. 4 shows how a set of concepts $\{x, y, r, s, t, w, z\}$ is partitioned into three branches: $\{x \to y \to r \to s,\ t \to w,\ z\}$, where $x \to y$ means that y must come after x in the curriculum. For each *branch*, there is no background knowledge or library at the beginning. The library grows as the teacher-learner protocol progresses in each branch.

Let $Q = \{c_i\}_{i=1}^{n}$ be a set of labelled concepts, a *curriculum* $\pi = \{\sigma_1, \cdots, \sigma_m\}$ is a full partition of Q, where each of the m subsets $\sigma_j \subset Q$ has a total order, being a sequence.

Definition 6 Let Q be a set of concepts. Let $\pi = \{\sigma_1, \sigma_2, \cdots, \sigma_m\}$ a curriculum in Q. We define the teaching size of each sequence $\sigma = \{c_1, c_2, ..., c_k\}$ as $TS_\ell(\sigma) = TS_\ell(c_1) + \sum_{j=2}^{k} TS_\ell(c_j | c_1, \ldots, c_{j-1})$. The overall teaching size of π is just $TS_\ell(\pi) = \sum_{i=1}^{m} TS_\ell(\sigma_i)$.

We denote $\overline{Q}$ as the set of all the possible curricula with Q. The order in which the subsets are chosen does not matter, but the order each subset is threaded does. For example, the curriculum $\pi = \{x \to y \to r \to s, t \to w, z\}$ has many paths, such as *xyrstwz* or *zxyrstw*. But note that π is different from $\pi' = \{y \to x \to r \to s, w \to t, z\}$.

The number of possible curricula given a number of concepts grows fast and this will motivate the heuristic we will introduce later on. In particular, the number of distinct curricula is given by the following calculation:

Proposition 1 *For any Q with n concepts, the number of different curricula is*

Table 2 Curricula for Example 1

Curriculum	Overall Teaching Size (bits)
$\pi_0 = \{a, b, c\}$	$TS(\pi_0) = TS(a) + TS(b) + TS(c) = 21 + 24 + 24 = 69$
$\pi_1 = \{a \to b \to c\}$	$TS(\pi_1) = TS(a) + TS(b\|a) + TS(c\|a, b) = 21 + 12 + 24 = \mathbf{57}$
$\pi_2 = \{b \to a \to c\}$	$TS(\pi_2) = TS(b) + TS(a\|b) + TS(c\|b, a) = 24 + 21 + 24 = 69$
$\pi_3 = \{c \to a \to b\}$	$TS(\pi_3) = TS(c) + TS(a\|c) + TS(b\|c, a) = 24 + 30 + 24 = 78$
$\pi_4 = \{c \to b \to a\}$	$TS(\pi_4) = TS(c) + TS(b\|c) + TS(a\|c, b) = 24 + 36 + 30 = \mathbf{90}$
$\pi_5 = \{a \to c \to b\}$	$TS(\pi_5) = TS(a) + TS(c\|a) + TS(b\|a, c) = 21 + 24 + 21 = 66$
$\pi_6 = \{b \to c \to a\}$	$TS(\pi_6) = TS(b) + TS(c\|b) + TS(a\|b, c) = 24 + 24 + 30 = 78$
$\pi_7 = \{a \to b, c\}$	$TS(\pi_7) = TS(a) + TS(b\|a) + TS(c) = 21 + 12 + 24 = \mathbf{57}$
$\pi_8 = \{b \to a, c\}$	$TS(\pi_8) = TS(b) + TS(a\|b) + TS(c) = 24 + 21 + 24 = 69$
$\pi_9 = \{a \to c, b\}$	$TS(\pi_9) = TS(a) + TS(c\|a) + TS(b) = 21 + 24 + 24 = 69$
$\pi_{10} = \{c \to a, b\}$	$TS(\pi_{10}) = TS(c) + TS(a\|c) + TS(b) = 24 + 30 + 24 = 78$
$\pi_{11} = \{b \to c, a\}$	$TS(\pi_{11}) = TS(b) + TS(c\|b) + TS(a) = 24 + 24 + 21 = 69$
$\pi_{12} = \{c \to b, a\}$	$TS(\pi_{12}) = TS(c) + TS(b\|c) + TS(a) = 24 + 36 + 21 = 81$

$$|\overline{Q}| = n! \cdot \left(\sum_{k=0}^{n-1} \binom{n-1}{k} \cdot \frac{1}{(k+1)!} \right) \tag{1}$$

Proof For any set $Q = \{c_1, \dots, c_n\}$ of n concepts, there are $n!$ permutations of n labelled elements. For each permutation, there are $n - 1$ possibilities of starting a *branch*. Consequently, we can choose k positions out of $n - 1$. This implies that there will be $k + 1$ subsets which can change its order, i.e., $(k + 1)!$ different permutations of the subsets express the same case.

Therefore, there are $n! \cdot \binom{n-1}{k} \cdot \frac{1}{(k+1)!}$ cases. Since $k \in \{0, 1, \dots, n - 1\}$, Eq. (1) gives the total number of distinct curricula.

□

Once that we know how many different curricula there are, we can try to identify which ones have lowest overall teaching size, denoted by TS^*_Q. A curriculum π is hence minimal if $TS(\pi) = TS^*_Q \leq TS(\pi'), \forall \pi' \in \overline{Q}$.

Regarding Example 1, there are thirteen distinct curricula in $\overline{Q}$ according to Proposition 1. Using Table 1, we can build Table 2, showing the overall teaching size for each curriculum in $\overline{Q}$.

Concerning all the teaching sizes taken as a whole and the descriptions measured in bits, Fig. 5 compares some relevant curricula from Table 2.

In terms of overall teaching size, there is a tie between π_1 and π_7, both being minimal (Fig. 5). However, we observe that there are big differences when considering overall teaching size for other curricula. For example, if we choose option π_4 instead of π_1 (or π_7), we get $\approx 63.3\%$ extra cost, even though there are just three concepts.

Further examples with more concepts and slightly different situations can be found in Section B.

From these examples, we ask ourselves whether it is necessary to calculate all the teaching sizes, as we have already done in Table 1 (or Tables 9 and 11 of Section B), to find

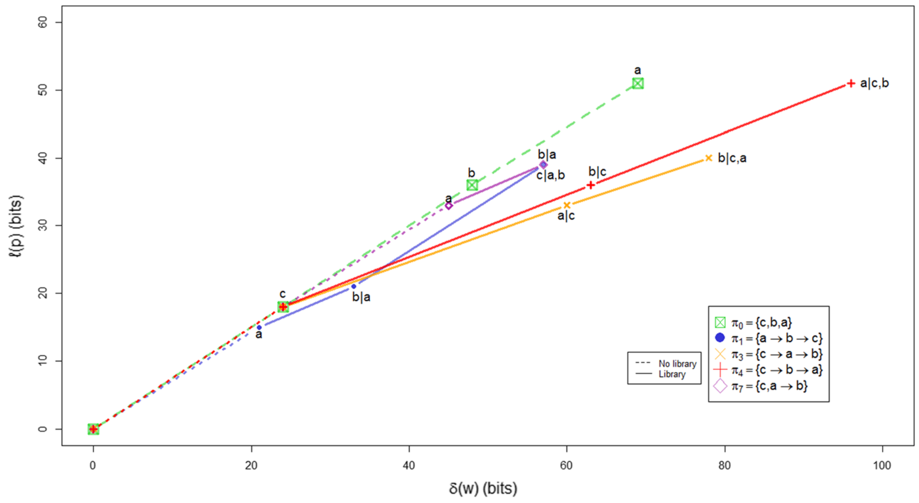

Fig. 5 Comparison of some relevant curricula of Example 1 (Table 1)

minimal curricula. The answer is negative, as we will see, but how can such a reduction of calculations be achieved? Moreover, is there a less costly procedure that could provide a close-to-optimal solution? We will deal with these issues in the following section.

5 Sufficient conditions to infer conditional teaching size

We first ask the question of whether it is possible to approximate the conditional teaching size of a concept c if the learner is given a new primitive for the library. Namely, given that $\Phi(w_c|B) = p_c \in [c]_L$, could it be possible to approximate $TS(c|\langle B, p\rangle)$, where p is a new primitive? It is also very important not only to identify a good approximation, but not to overestimate the real conditional teaching size. We will employ this conservative property (in Sect. 6) to define an algorithm that outputs minimal curricula.

Firstly, we give a sufficient condition, valid for any kind of language, either universal or not, that provides an underestimation of conditional teaching size. Secondly, we identify when such sufficient condition applies for our particular drawing domain.

The following corollary provides that sufficient condition. It is proved in Section D as a consequence of Lemma 4.

Corollary 2 *Let w_c, $w'_c \subset X$ such that $p_c = \Phi_\ell(w_c|B)$ and $p'_c = \Phi_\ell(w'_c|B')$, where B' is a library that extends, with a new primitive, the library B. Let us suppose that the following conditions are met:*

I $\exists k_1, k_2 \in \mathbb{N}$ *such that* $\delta(w_c) = \ell(p_c) + k_1$ *and* $\delta(w'_c) = \ell(p'_c) + k_2$
II $\delta(w_c) - \delta(w'_c) \leq \ell(p_c) - \ell(p'_c)$
III $\ell(p'_c) \leq \ell(p_c)$

Then,

$$\frac{\ell(p'_c)}{\ell(p_c)} \cdot \delta(w_c) \leq \delta(w'_c) \tag{2}$$

It is important to note that Inequality (2) does not overestimate $\delta(w'_c)$; this fact will be key in Sect. 6 to define an algorithm that ouputs optimal curricula.

With respect to our drawing domain, Condition I (Corollary 2) is always true, since the language L has equivalent instructions for every command in Σ. Also, the concepts of the drawing domain meet Condition II (Corollary 2), as a result of Theorem 3 (Section C), when w_c has no negative examples. Otherwise, let us suppose that $w_c = \{e^+, (e^-_i)\}$, such that $\exists p_i$ in L_B with $p_i \vDash e^+_i$ and $p_i \notin [c]_L$. In general, we could not assure that adding a new primitive to B' would make $\Phi(w'_c|B') \prec p_i$, $\forall i$. If so, it would be unnecessary to include negative examples in w'_c, and Condition II would not be met. However, if concepts can be taught initially in language L ($|B| = 0$) without negative examples, then we can always estimate successively the teaching size of such concepts. In other words, we will consider $\frac{\ell(p'_c)}{\ell(p_c)} \cdot TS(c) \leq TS(c|B')$ (instead of $TS(c|B)$ on the left side of the inequality).

Finally, we assumed in Condition III (Corollary 2) that the length in bits of a library call, @i, is the same as the new call @i′. But there exist cases such as $\Phi(\{\text{ENESEE}^+\}|\langle\text{RD}\rangle\rangle) = \text{R)U@} \in [c]_L$ and $\Phi(\{\text{ENESEE}^+\}|\langle\text{RD}\rangle, \text{RURD}\rangle\rangle) = \text{R)U@0} \in [c]_L$. In such cases, $\frac{\ell(p^*_{c|B'})}{\ell(p^*_{c|B})} > 1$ and Inequality (2) is not ensured, where $p^*_{c|B}$ is the first expression in language L enhanced with library B, using order $\prec$, that identifies concept c[2] (we denote $\ell(p^*_{c|B}) = \ell(p^*_c)$ when $|B| = 0$).

However, in those situations ($\ell(p^*_{c|B'}) > \ell(p^*_{c|B})$), we consider $\delta(w_c) = \delta(w'_c)$, since $\delta(w'_c)$ cannot reduce $\delta(w_c)$.

These considerations lead us to the definition of a valid family of heuristics that always output optimal curricula without calculating all the teaching sizes.

6 Heuristic search for optimal curricula

In our case, the search space is given by all the curricula, $\overline{Q}$, and we need to find at least one π such that $TS(\pi) = TS^*_{\overline{Q}}$. Each internal *node* in the *search graph* is a partial curriculum (not covering all concepts), while each *leaf* (node with no children at the bottom level) is a full curriculum belonging to $\overline{Q}$, and an edge means adding a new concept to a branch of the curriculum (remember a curriculum is actually a tree). For instance, for two concepts a and b, the root n_1 would be the empty curriculum. The children at level 2 would be $n_{1,1} = \{a\}$ and $n_{1,2} = \{b\}$. Finally, at level 3, the children of $n_{1,1}$ would be $n_{1,1,1} = \{a \rightarrow b\}$, $n_{1,1,2} = \{a, b\}$ and the children of $n_{1,2}$ would be $n_{1,2,1} = \{b \rightarrow a\}$, $n_{1,2,2} = \{b, a\}$, which means that we have two nodes that would be equal, and we see that the search space is a directed acyclic graph.

Let us start with a simple graph traversal algorithm and then evolve it into more sophisticated procedures. The standard A^* search is a baseline graph traversal algorithm Russell and Norvig (2020). A^* is based on a node evaluation function

[2] The conditional Kolmogorov complexity of concept c given the library B is defined as $K_L(c|B) = \ell(p^*_{c|B})$.

$$f(n) = g(n) + h(n), \tag{3}$$

where $g(n)$ is the overall cost of getting to node n from the beginning and $h(n)$ is the *heuristic* function, i.e., the estimated cost of getting to a *target node* (leaf node) from node n. A^* search guarantees an optimal solution when the heuristic function meets the following conditions:

1. $h(n) \geq 0$, for every node n.
2. $h(n) = 0$, if n is a *target* node.
3. $h(n)$ is *admissible*, i.e., $h(n)$ never overestimates.

A popular variant of A^* is *Weighted* A^*_α (WA*), another graph traversal algorithm that guides the search with $h'(n) = \alpha \cdot h(n)$, where α is a parameter. Remember that if $h(n)$ is admissible then WA* guarantees the optimal solution when $0 < \alpha \leq 1$. Sometimes, it is useful to employ WA*, because it often gives a close-to-optimal solution with less node expansions (Hansen & Zhou, 2007).

Let us look for admissible heuristics not only because they guarantee success for algorithm A^*, but also because they help to assess other close-to-optimal solutions (Hansen & Zhou, 2007).

Definition 7 (Node cost) Let Q be a set of concepts and n be a node of the search space $\overline{Q}$. We define the teaching size cost of node n, which we denote by $g(n)$, as the overall teaching size of the curriculum, either partial or complete, that represents node n.

Note that any path of the search graph uniquely generates a library B as background knowledge. For instance, the path $(a \rightarrow b \rightarrow c)$ generates library $B = \langle p^*_a, p^*_{b|\langle p_a \rangle} \rangle$ when reaching concept c, while the path $(a \rightarrow b, c)$ generates no prior knowledge, i.e., $|B| = 0$, when getting to concept c.

We now proceed with the theoretical results that will allow us to define an admissible heuristic. But, firstly, we need to define the cost of *crossing an edge* of the search graph.

Definition 8 (Estimated edge cost) Let Q be a set of concepts and n be a node of the search space $\overline{Q}$; the last edge of node n is e. Let concepts a and b be the vertices of edge e, either $\mathsf{e} = [a \rightarrow b]$ or $\mathsf{e} = [a, b]$. Let B be the library employed to reach concept b through the edge e, where $0 \leq |B|$. We define the *estimated cost of crossing* e as

$$h_B(\mathsf{e}) = \frac{\ell(p^*_{b|B})}{\ell(p^*_b)} \cdot TS(b)$$

If $|B| = 0$ then $\ell(p^*_{b|B}) = \ell(p^*_b)$ and $h_B(\mathsf{e}) = TS(b)$.

We note that it would also be possible to define the estimated edge cost using the new primitive employed between one node and its child. That is to say, if B is the library employed to reach a, and B' is the library used to get to b, then we could define

$$h_{B' \setminus B}(\mathsf{e}) = \begin{cases} \frac{\ell(p^*_{b|B'})}{\ell(p^*_{b|B})} \cdot TS(b), & \text{if } \ell(p^*_{b|B'}) \leq \ell(p^*_{b|B}) \\ TS(b), & \text{otherwise} \end{cases}$$

Note that $h_B(\mathsf{e}) \leq h_{B'\setminus B}(\mathsf{e}), \forall \mathsf{e}$. Consequently, $h_B(\mathsf{e})$ is less dominant than $h_{B'\setminus B}(\mathsf{e})$. Accordingly, if the former reduces the computational cost to identify optimal curricula, the latter will perform even better.

We now extend Definition 8 to a path of the search graph.

Definition 9 Let n, m be nodes of the search space $\overline{Q}$, where m is a child of n that might be several levels after n. Let $\{\mathsf{e}_i\}$ be a path from node n to node m; as a special notation case we use $\{\mathsf{e}_i\} = \{\emptyset\}$ when there is no path between n and m throughout the search space. Let B_i, with $0 \leq |B_i|, \forall i$, be the library employed to cross the edge e_i. We define the *estimated cost* of *getting* from node n to node m as

$$H_n(m) = \begin{cases} \infty, & \text{if } \{\mathsf{e}_i\} = \{\emptyset\} \\ \sum_i h_{B_i}(\mathsf{e}_i), & \text{otherwise} \end{cases}$$

We now employ Definition 9 to define the estimated cost of a node.

Definition 10 (Estimated cost of node n) Let n be a node of $\overline{Q}$, we define the estimated cost of node n as

$$\mathcal{H}(n) = min\{H_n(m) : m \textit{ is a leaf node of the search graph}\}$$

The leaf nodes considered in Definition 10 are any leaf node of the search graph, not only the ones that are descendants of a given node n. The following Examples 2 and 3 illustrate how to calculate estimated costs in particular situations.

Example 2 Let a and b be the concepts of Example 1; where $p_a^* = \mathsf{RURD)}$ and $p_b^* = \mathsf{RRURD)}$. We consider the node $\{a\}$ in the search space whose leaf nodes are $\{a, b\}$, $\{b \to a\}$ and $\{a \to b\}$. We want to calculate $\mathcal{H}(\{a\})$; since *we are* already in node $\{a\}$ the calculations are

$$\begin{aligned} &H_{\{a\}}(\{a, b\}) = TS(b) = 24, \\ &H_{\{a\}}(\{b \to a\}) = \infty \text{ and} \\ &H_{\{a\}}(\{a \to b\}) = h_{\langle p_a \rangle}([a \to b]) = \frac{\ell(p^*_{b|\langle p_a^* \rangle})}{\ell(p_b^*)} \cdot TS(b) = \frac{\ell(\mathsf{R@})}{\ell(\mathsf{RRURD)})} \cdot 24 = 8. \end{aligned}$$

Therefore, $\mathcal{H}(\{a\}) = min\{24, \infty, 8\} = 8$.

Example 3 Let a, b and c be the concepts of Example 1, where $p_c^* = \mathsf{R)URD)}$. The leaf nodes with bounded estimation cost from node $\{a\}$ are $\{a, b, c\}$, $\{a \to b, c\}$, $\{a \to c, b\}$, $\{a \to b \to c\}$ and $\{a \to c \to b\}$. For instance, the estimated cost of node $\{a \to b, c\}$ from node $\{a\}$ is

$$H_{\{a\}}(\{a \to b, c\}) = h_{\langle p_a^* \rangle}([a \to b]) + h_{\emptyset}([b, c]) = \frac{\ell(p^*_{b|\langle p_a^* \rangle})}{\ell(p_b^*)} \cdot TS(b) + TS(c) = 8 + 24 = 32.$$

Similarly, $H_{\{a\}}(\{a \to c, b\}) = h_{\langle p_a^* \rangle}([a \to c]) + h_{\emptyset}([c, b]) = \frac{\ell(p^*_{c|\langle p_a^* \rangle})}{\ell(p_c^*)} \cdot TS(c) + TS(b) = 1 \cdot 24 + 24 = 48$,

$$\begin{aligned}
H_{\{a\}}(\{a \rightarrow b \rightarrow c\}) =& h_{\langle p_a^* \rangle}([a \rightarrow b]) + h_{\langle p_a^*, p_b^* \rangle}([b \rightarrow c]) \\
=& \frac{\ell(p^*_{b|\langle p_a^* \rangle})}{\ell(p_b^*)} \cdot TS(b) + \frac{\ell(p^*_{c|\langle p_a^*, p_b^* \rangle})}{\ell(p_c^*)} \cdot TS(c) = \frac{\ell(\mathsf{R@})}{\ell(\mathsf{RRURD}))} \cdot 24 + 1 \cdot 24 = 32, \\
H_{\{a\}}(\{a \rightarrow c \rightarrow b\}) =& h_{\langle p_a^* \rangle}([a \rightarrow c]) + h_{\langle p_a^*, p_c^* \rangle}([c \rightarrow b]) \\
=& \frac{\ell(p^*_{c|\langle p_a^* \rangle})}{\ell(p_c^*)} \cdot TS(c) + \frac{\ell(p^*_{b|\langle p_a^*, p_c^* \rangle})}{\ell(p_b^*)} \cdot TS(b) \\
=& 1 \cdot 24 + \frac{\ell(\mathsf{R@0})}{\ell(\mathsf{R)URD}))} \cdot 24 = 33.\dot{3} \text{ and } H_{\{a\}}(\{a, b, c\}) = 69.
\end{aligned}$$

Therefore, $\mathcal{H}(\{a\}) = min\{\infty, 69, 48, 32, 33.\dot{3}\} = 32$.

The heuristic of Definition 10 never overestimates the teaching size of a node if conditions expressed in Theorem 3 apply. As a result, A^* will output minimal curricula if we define the heuristic $h(n) = \mathcal{H}(n)$. Let us show some experiments.

7 Empirical results

We experimented with three sets of concepts: $Q = \{a, b, c\}$ (Example 1 in Sect. 3), $Q' = \{a', b', c\}$ and $Q'' = \{a, b, c, d\}$ (Examples 4 and 5, respectively, in Section B). We studied all the possible different curricula for each set of concepts. Thus, we had to calculate all the teaching sizes of Tables 1, 9 and 11. Calculations took a long time even that we utilised a HPCx cluster[3]. It was a server shared with other users, but there were reserved three cores for every teaching size calculation. Namely, it took approximately nine days to compute $TS(c|a)$ (Table 1) and when the library had more primitives then the calculations increased explosively. For instance, it took more than two months to calculate $TS(c|a, b, d)$ (Table 11).

As we mentioned before, the admissible heuristic $h(n) = \mathcal{H}(n)$ is valid for every set of concepts Q of our drawing domain, when each concept $c \in Q$ can be taught in L ($|B| = 0$) without negative examples. We implemented the A^* search using such heuristic and applied it to the examples.

In Example 1 the algorithm finds a minimal curriculum π_7 in 4 steps, through 8 effective calculations of teaching size against the overall 15 edges. Regarding Example 4, A^* finds the minimal curriculum in 4 steps, effectively calculating 7 teaching sizes against 15 overall.

Considering Example 5 (Section B), there are two curricula that maximise the teaching size with, approximately, 66% more effort than the optimal. The A^* search showed $\pi^* = \{d \rightarrow a \rightarrow b \rightarrow c\}$ as minimal curriculum in 6 steps, through 13 teaching size calculations against 64 (20% teaching size calculations). Experiments show that there is only one optimal curriculum ($TS(\pi^*) = 63$), and it coincides with the one that identifies the A^* search.

Tables 3, 4 and 5 summarise the experimental results obtained for Examples 1, 4 and 5, respectively, using the following distinct algorithms:

[3] https://www.uclm.es/Areas/AreaTIC/Servicios/Investigacion/SSC.

Table 3 Curricula obtained for Example 1 through different algorithms and whether they are optimal (in boldface when it has to be optimal), the % of TS operations that need to be calculated (the lower the better) and the % of extra cost when it is not optimal

$Q = \{a, b, c\}$	D	D'	G	$A^*_{\alpha=1}$	$A^*_{\alpha=0.8}$	$A^*_{\alpha=1.5}$
π	$a \to b, c$	$a \to b, c$	$a \to b, c$	$a \to b, c$	$\frac{a \to b, c}{a \to b \to c}$	$a \to c \to b$
Optimal?	**Yes**	Yes	Yes	**Yes**	**Yes**	No
% TS calcul	93.3%	66.6%	40%	53.3%	60%	40%
% Extra TS	0%	0%	0%	0%	0%	15.8%

Table 4 Curricula obtained for Example 4 through different algorithms and whether they are optimal (in boldface when it has to be optimal), and the % of TS operations that need to be calculated (the lower the better)

$Q' = \{a', b', c\}$	D	D'	G	$A^*_{\alpha=1}$	$A^*_{\alpha=0.8}$	$A^*_{\alpha=1.5}$
π	$a' \to b', c$	$a' \to b', c$	$a' \to b', c$	$a' \to b', c$	$a' \to b', c$	$a' \to b', c$
Optimal?	**Yes**	Yes	Yes	**Yes**	**Yes**	Yes
% TS calcul	93.3%	66.6%	40%	46.6%	53.3%	40%

Table 5 Curricula obtained for Example 5 through different algorithms and whether they are optimal (in boldface when it has to be optimal), the % of TS operations that need to be calculated (the lower the better) and the % of extra cost when it is not optimal

$Q'' = \{a, b, c, d\}$	D	D'	G	$A^*_{\alpha=1}$	$A^*_{\alpha=0.8}$	$A^*_{\alpha=1.5}$
π	$d \to a \to b \to c$	$d \to a \to b, c$	$d \to a \to b \to c$	$d \to a \to b \to c$	$d \to a \to b \to c$	$d \to a \to b, c$
Optimal?	**Yes**	No	Yes	**Yes**	**Yes**	No
% T.S. calc	84.4%	46.9%	15.6%	23.4%	39.1%	15.6%
% Extra TS	0%	4.8%	0%	0%	0%	4.8%

- Dijkstra's algorithm (D) Dijkstra (1959). It always identifies an optimal curriculum, since teaching sizes are positive Barbehenn (1998).
- Dijkstra's modified algorithm (D'): it stops when it goes through all the concepts. It does not necessarily get an optimal curriculum, but the computational cost is lower than Dijkstra's algorithm in terms of teaching size calculations.
- Greedy algorithm (G): from any given state, it always chooses the bifurcation that involves the least teaching size. In general, it does not identify an optimal curriculum.
- A^* search algorithm Hart et al. (1968). The heuristic given in Definition 10, $\mathcal{H}$, guarantees an optimal curriculum.

- A^*_α search, i.e., WA* algorithm with parameter α: since the heuristic $\mathcal{H}(n)$ is admissible, it will identify optimal solutions when $0 \leq \alpha \leq 1$[4].

As we can see in Tables 3, 4 and 5, there are big differences in terms of computational costs when using procedures ensuring optimal teaching curricula as an output. For instance, in Example 1, the computational cost of D performs an extra 40% with regard to $A^*_{\alpha=1}$ (33.3% when $\alpha = 0.8$); in Example 4 the comparative between D and $A^*_{\alpha=1}$ shows a 46.7% extra cost for D (40% when $\alpha = 0.8$). Furthermore, when we increase the number of concepts involved, as in Example 5, $A^*_{\alpha=1}$ makes 61% less computing than D. It is reasonable to think that the tendency might double the computational reduction of A^* with respect to A^* when we add a new concept.

It is true that the greedy algorithm G identifies an optimal curriculum in all the Examples (1, 4 and 5), with even less computational effort. However, such procedure is risky, since an optimality is not guaranteed then and the difference in TS-calculations % is 13.3% in Example 1, 6.6% in Example 4 and 7.8% in Example 5, which is not so high with respect to the option ensuring optimal teaching curricula $A^*_{\alpha=1}$.

8 Discussion

In this paper, we provided sufficient conditions to define a procedure that effectively identifies optimal curricula. This is used in an A^* search enhanced by the heuristic given in Definition 10, for a given set of concepts.

Inequality (2) can be applied to similar machine-teaching settings using different languages L, either universal or not. It is sufficient to consider a similar machine-teaching scenario and a set of concepts that meet the conditions.

Thus, we may introduce new instructions such as '|' (logical OR operator), and even discard instructions like ')'. In particular, the previous results, dedicated to obtaining a heuristic, can be applied to any language L' that meets these statements:

(i) L' should contain equivalent instructions for every command of Σ.
(ii) The size in bits of the equivalent instructions in L' should be less or equal than their counterpart commands in Σ.
(iii) There is an instruction, similar to @, that is able to retrieve primitives in language L' and is the last instruction considering the lexicographical order.

We have shown that it is possible to apply a heuristic search to those MT scenarios that satisfy the above conditions (i), (ii) and (iii), having the following properties:

1. For all the cases studies shown, it provides a reduction of the computational effort involved.

[4] Note that A^* search is useful when we do know a heuristic, which is one reason why identifying a good heuristic is so important. Whether we use WA* or A^*, it is key to know an admissible heuristic. Anytime A^*, a version of WA* algorithm with weight α gradually reducing until $\alpha = 1$, can use non-admissible heuristics, but knowing $h(n)$, together with an upper bound on the optimal solution, helps pruning the search space and detects convergence to an optimal solution Hansen and Zhou (2007).

2. The identification of optimal curricula is guaranteed.

Our approach is defined for compositional languages of discrete character and, correspondingly, a discrete optimisation process over a discrete space. As a result, it is not straightforward to adapt the setting to continuous representations, including neural networks. However, the general framework may adapt to a language L that might be based on continuous principles: instructions in L might be given different weights, so that the total order $\prec$ might be defined through a distribution of probabilities. However, our MT framework is more indicated in situations where we want to understand –and explain– how each concept builds on previous concepts compositionally. For instance, given an observation, what is the best explanation for that information? In fact, programs in the language L of our particular 2D drawing domain (Section A) can be described in terms of automata, which have also been utilised recently in AI to spot patterns describing data sets in 2D too (Das et al., 2023). We can see the discrete character of our approach and the language we have used as a limitation, but it can also be seen as an opportunity for other researchers to investigate the performance of our heuristic search in other MT settings (even continuous) and domains.

Overall, our starting point was the realisation that curriculum learning for a given set of compositional concepts is underdeveloped mainly because of computational costs. Thanks to the new heuristics introduced in this paper, we are now able to effectively implement CL in MT, not only at the level of improving a sequence of training examples for a given task, but also considering the combinatorial explosion of interlocking concepts.

Appendix A: The drawing domain

In this section we establish the setting used by the teacher to generate examples and the language employed by the learner to interpret them.

A.1. The space of examples: polygonal chains

Given a point, consider that we draw a *polygonal chain*, using 2D axis-parallel unit segments from that initial point, such that the end of a segment coincides with the beginning of another segment (hence the term chain). We use the *unit strokes* North, South, East and West as *commands* (Sect. 2), to define a quaternary alphabet $\Sigma_4 = \{\mathsf{N}, \mathsf{S}, \mathsf{E}, \mathsf{W}\}$. Note that different chains may lead to the same *shape*.

We consider Σ_4^*, which is formed by all possible concatenations of zero or more symbols in Σ_4; note that it also includes the *empty string*, which is denoted by ε. We define an order $\lessdot$ over Σ_4^* by first considering the size and, for ties, the lexicographical order: N, S, E and W.

For example, in Fig. 6, starting at the black dot, the square in the upper left corner can be described by the chains ENWS, NESW or even NESWNE. The only condition is that such sequences must be built with a single drawing, i.e., the pen cannot be lifted from the paper until the end. For instance, to draw a T-shape (second drawing on the top of Fig. 6), we need four strokes at least, NWEE or NEWW, even though the shape only has three segments.

We label such examples as *positive* (+) and *negative* (−).

There are six symbols to produce examples: N, S, E, W, + and −; the last two indicate whether the example is positive or negative. We will use $\lceil log_2(6) \rceil = 3$ binary digits

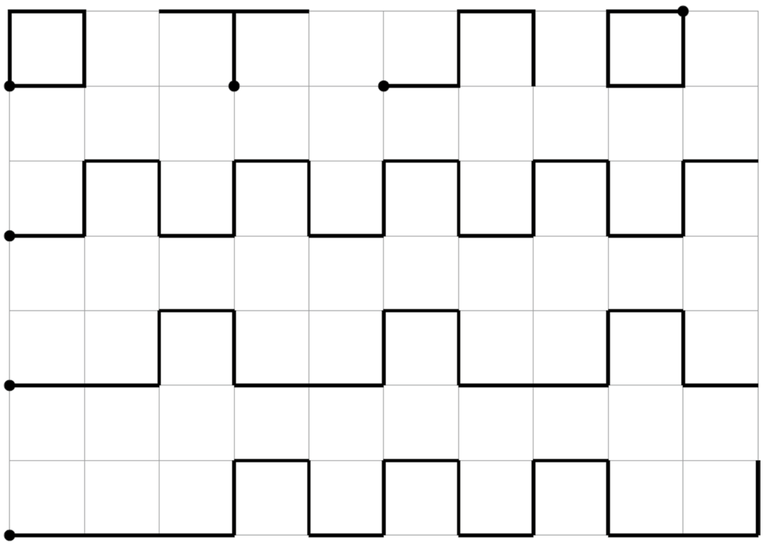

Fig. 6 Different shapes derived from polygonal chains

to express each symbol.[5] Note that + and − are useful for the teacher and the learner when exchanging sets of examples, not only to express which ones are positive or negative, but also to identify when an example terminates and a new one begins. The size in bits of an example, denoted by $\delta(x)$, either positive or negative, is the length in bits of its representation. For example, $\delta(\mathsf{NESW}^+) = 15$ bits.

We can also extend the total order $\lessdot$ to X by first considering the size and, for ties, the lexicographical order: N, S, E, W, + and −.

The infinite concept class C is the set of all subsets of X. For any concept $c \in C$ the teacher's objective is to find the shortest set of examples from which the learner will precisely distinguish the concept.

A.2. Language syntax (instructions) and semantics (automata)

We now define a language that is employed by the learner to identify concepts from a set of examples provided by the teacher. Let us consider the *instruction* alphabet $L_6 = \{\mathsf{U}, \mathsf{D}, \mathsf{R}, \mathsf{L},), @\}$ to define the syntax of the programs of the language $L = L_6^*$ as the set of programs that the teacher and the learner use to think of and identify concepts. The meaning of the symbols is already given in Sect. 2.

We denote the number of bits of p in language L as $\ell(p)$; $\dot{\ell}(p)$ denotes its number of instructions. Since the alphabet size $|L_6| = 6$, we will have a total of 6 symbols that will need 3 bits for each instruction. For instance, in our setting, $\ell(\mathsf{URRRD}) = 15$ bits.

Regarding the implementation of *prior knowledge*, we consider that the language works with a library of *primitives*, which is simply a possibly empty ordered set of programs $B = \langle p_1, p_2, ... \rangle$, with p_j in L. The instruction @ represents a call for the library B, when

[5] There are more efficient codings, but by using + and − as extra symbols, we can also use them as delimiters. We use 3 bits for each character instead of $log_2(6)$, as this only affects size by a multiplicative factor.

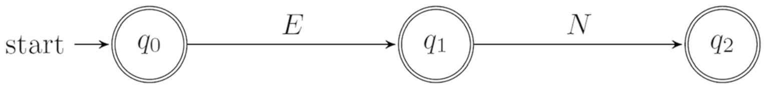

Fig. 7 Automaton built from expression RU

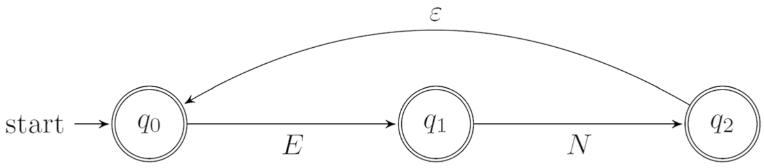

Fig. 8 Automaton identified by the expression RU)

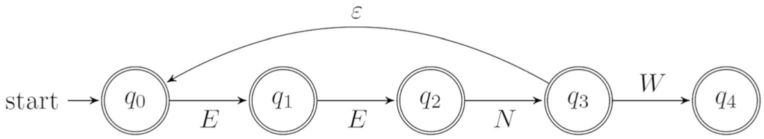

Fig. 9 Automaton identified by the expression RRU)L

there is just one primitive. The expression @i calls the primitive indexed with a binary string i representing natural number $j-1$, where j is the position in the library. For example, R@, is equivalent to RD) when $B = \langle \mathsf{D}) \rangle$. For instance, the library $B = \langle \mathsf{R}, \mathsf{RR}, \mathsf{RRR} \rangle$ makes the expressions U@10D and URRRD as it is calling the third primitive (indexed by 10). Since the dimension of the library is already known by the learner and all the instructions can be translated into binary strings, we consider that it is not necessary to explicitly add the two binary digits for composing binary strings to index library calls. In the end, any expression in language L is uniquely translated into a binary string. We denote L_B as the language L that employs library B.

The size in bits of a call to the library is $\ell(@\mathsf{i}) = \ell(@) + \lceil \log_2(|B|) \rceil$ bits. In this way, $\ell(\mathsf{U@01D}) = 11$ bits, i.e., four bits less than the equivalent program URRRD when using library $B = \langle \mathsf{R}, \mathsf{RR}, \mathsf{RRR} \rangle$. Note that binary representations utilise trailing zeros, e.g., 00 or 01. We observe that, every chain in language L can be uniquely translated into L_5^*, where $L_5 = L_6 \backslash \{ @ \}$, by substituting the corresponding primitive for its corresponding library call.

There is a total order, $\prec$, over the language L_B defined by two criteria: (i) length and (ii) lexicographic order, $\mathsf{U} \prec \mathsf{D} \prec \mathsf{R} \prec \mathsf{L} \prec) \prec @$, only applied when two expressions have equal size; when considering library calls, 0 is lexicographically previous to 1, e.g., U@00D $\prec$ U@01D.

We will consider that a command of Σ_4 is *equivalent* to a single instruction of the visual language when they represent the same phenomenon. For instance, the unit stroke N of X is equivalent to instruction U of L, since both represent equal moves on the grid.

In our particular setting, we will employ the function $\phi : L_6 \rightarrow \Sigma_4$, such that $\phi(\mathsf{U}) = \mathsf{N}$, $\phi(\mathsf{D}) = \mathsf{S}$, $\phi(\mathsf{R}) = \mathsf{E}$ and $\phi(\mathsf{L}) = \mathsf{W}$ to identify stroke-instruction equivalences.

To complete the representation mapping between L_6 and Σ_4, i.e., the semantics of L, we will uniquely associate a *deterministic finite automaton*, *DFA* (McCulloch & Pitts,

Table 6 Transition table for RD)

Entry states	E	S	ε
q_0	q_1	$\emptyset$	$\emptyset$
q_1	$\emptyset$	q_2	$\emptyset$
q_2	$\emptyset$	$\emptyset$	q_0

Table 7 First stage of the translation of R)URD) into an automaton

Instructions	α_1	β_1	α_2	α_3	α_4	β_4
q_0	q_1	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$
q_1	$\emptyset$	q_0	q_2	$\emptyset$	$\emptyset$	$\emptyset$
q_2	$\emptyset$	$\emptyset$	$\emptyset$	q_3	$\emptyset$	$\emptyset$
q_3	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	q_4	$\emptyset$
q_4	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	q_0

1943), for each program p in language L. In our particular setting, an automaton represents the *semantics* of a program p in L. In other words, the set of examples S the program covers (or can generate) is equal to the set of strings that the corresponding automaton can accept.

For example, the program RU in language L uniquely represents the automaton of Fig. 7. Such automaton recognises the concept $\{E^+, EN^+\}$. RU also recognises $\{E^+\}$, a behaviour similar to some spellcheckers. Among other tools, automata are employed in AI to capture latent patterns (Das et al., 2023).

Regarding the instruction), for instance, Figs. 8 and 9 represent the automata RU) and RRU)L, respectively. Note that the automaton RU) accepts ENE, since we can employ the empty string ε to follow), i.e., ENεE.

Specifically, we give the following procedure to uniquely associate an automaton to a program in L. But firstly, we will restrict such association to L_5^*, where $L_5 = L_6 \backslash \{@\}$. As we already mentioned, every program in language L can be translated into L_5^* by replacing the library calls. Also, for simplification purposes, we will not consider) redundancies, i.e., we substitute any)) occurrences by).

For instance, let us consider the expression R@) in the visual language L, when $B = \langle D) \rangle$. If we eliminate)) redundancies, then R@) is uniquely translated into L_5^* as RD). Thus, we get the DFA expressed by the transition diagram of Table 6, where all the states are of acceptance.

However, there can be more complex programs. For example, let us consider another program like R))URD). Firstly, we eliminate redundancies for instruction), so that we get R)URD). Then we parse the latter expression to distinguish instructions equivalent to commands in Σ_4:

- We denote $\sigma_1 = \alpha_1 =$ R as the first ϕ-equivalent instruction.
- We denote $\sigma_2 = \beta_1 =$), as the first instruction). Since there is only one previous ϕ-equivalent instruction, we use the subindex 1 for β.
- Similarly, we continue parsing the program as $\sigma_3 = \alpha_2 =$ U, $\sigma_4 = \alpha_3 =$ R, $\sigma_5 = \alpha_4 =$ D and $\sigma_6 = \beta_4 =$).

Table 8 Transition table for R)URD)

Entry states	E	ε	N	S
q_0	q_1	$\emptyset$	$\emptyset$	$\emptyset$
q_1	$\emptyset$	q_0	q_2	$\emptyset$
q_2	q_3	$\emptyset$	$\emptyset$	$\emptyset$
q_3	$\emptyset$	$\emptyset$	$\emptyset$	q_4
q_4	$\emptyset$	q_0	$\emptyset$	$\emptyset$

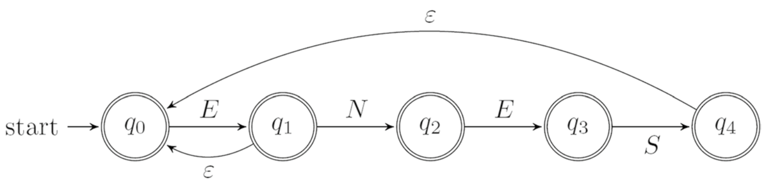

Fig. 10 Automaton identified by the expression R)URD)

Note that there are $m = 4$ ϕ-equivalent instructions in p and $n - m = 2$ instructions). Now we are able to build the DFA automaton $M = (\Theta, \Xi, s, F, \delta)$ where:

- $\Theta = \{q_0, q_1, q_2, q_3, q_4\}$ is the set of states.
- $\Xi = \{\phi(\alpha_1) = \mathsf{E}, \phi(\beta_1) = \varepsilon, \phi(\alpha_2) = \mathsf{N}, \phi(\alpha_3) = \mathsf{E}, \phi(\alpha_4) = \mathsf{S}, \phi(\beta_4) = \varepsilon\}$ is the set of input symbols.
- The start state is $s = q_0$.
- The set of accept states is $F = \Theta$.
- We now consider the function $\delta' : \Theta \times \{\sigma_i\}_{i=1}^n \rightarrow \Theta$, given by Table 7.
- Using δ', we define the transition function through the translation of $\{\sigma_i\}_{i=1}^n$ into Ξ, using $\phi : L_5 \rightarrow \Sigma_4 \cup \{\varepsilon\}$, as

$$\delta(q, \phi(\sigma_i)) = \delta'(q, \sigma_i), \forall q \in \Theta, \forall i$$

, which is expressed in Table 8.

Now, we are able to draw the graph associated to the program R)URD), represented in Fig. 10.

We can generalise the process that uniquely identifies the semantics of a program as an automaton through Definition 11.

Definition 11 Let $p = \sigma_1 \cdots \sigma_n \in L_5^*$ be a program such that:

- $\nexists i \in \{1, \dots, n-1\}$ with $\sigma_i = \sigma_{i+1} =$).
- We parse p from beginning to end and with α_j we denote every σ_i such that $\sigma_i \neq$), where j increases succesively from 0 to m, with $m \leq n$ being the number of instructions of p different from).
- Such parsing of p also makes β_k denote every σ_i such that $\sigma_i =$), where k is the number of instructions previous to σ_i different from).

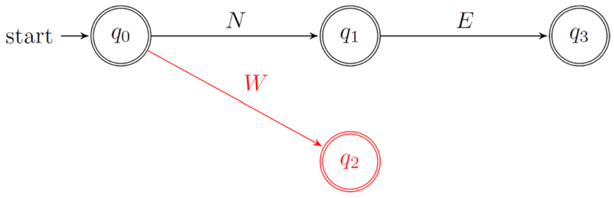

Fig. 11 The visual language L does not allow to build the edges q_0q_1 and q_0q_2 simultaneously

– We will extend $\phi : L_5 \rightarrow \Sigma_4 \cup \{\epsilon\}$ by considering $\phi()) = \epsilon$. Thus, we associate p with the DFA denoted as $M = (Q, \Xi, s, F, \delta)$, where:

1. The set of states is $\Theta = \{q_0, \dots, q_m\}$
2. The set of input symbols is $\Xi = \{\phi(\sigma_i)\}_{i=1}^n$
3. The start state is $s = q_0$
4. The set of accept states is $F = \Theta$
5. The transition function is $\delta : \Theta \times \Xi \rightarrow \Theta$, with

5.1) $\delta(q_r, \phi(\alpha_{r+1})) = q_{r+1}, \forall r \in \{1, \dots, m-1\}$ (and $\emptyset$ otherwise)
5.2) $\delta(q_r, \phi(\beta_r)) = q_0, \forall r \in \{0, \dots, m\}$ (and $\emptyset$ otherwise)

As a result, we identify a program p in language L with the DFA that it defines. Now, we say that program p in L *identifies* a concept $c \in 2^X$, and we denote $p \in [c]_L$, if the DFA defined by p recognises c. For example, RU $\in [\{\text{E}^+, \text{EN}^+\}]_L$.

We denote $[c]_L$, or simply $[c]$ when L is clear from the context, as the *equivalence class* of programs in language L that identify the same concept c. For example, RU)R and RU) identify the same concept, so they are equivalent.

We say that the program p in the visual language L *satisfies* a positive example, e^+, if the DFA defined by p accepts the word $e \in \Sigma_4^*$. For instance, RU) satisfies ENE$^+$. Regarding negative examples, a program satisfies a negative example, e^-, if it does not satisfy its positive counterpart, e^+. For instance, since RD) does not satisfy EN$^+$, then it satisfies EN$^-$. Conversely, R) satisfies EEE$^+$, but it does not satisfy EEE$^-$.

Given a set of examples $w \subset X$, a program satisfies w if it satisfies each example of w. For instance, RURD) satisfies the example set $\{\text{N}^-, \text{ENESEN}^+\}$.

Given an example e^+ with n unit strokes, we can restrict the search for the automaton that satisfies e^+, by generating up to $n+1$ states maximum. In other words, we use the complexity function $\mathsf{f}(n) = n + 1$ as an upper bound of the necessary states. For instance, the example EEN$^+$, with $n = 3$ unit strokes, is satisfied by a DFA represented by the program RRU using four states ($4 \leq \mathsf{f}(3)$), or even R)U, with just three states. However, given the example EEN$^+$, we will not consider automata like RRUU or RRUUU, because their number of states are higher than the bound given by the complexity function f.

We can see that in general the maximum number of states considered to test whether an automaton p satisfies a given positive example e^+ with n unit *strokes* is given by a complexity function $\mathsf{f}(n)$, such that $n \leq \mathsf{f}(n), \forall n$. Thus, we say that p is f–compatible with example e^+, and we denote $p \vDash_{\mathsf{f}} e^+$ (otherwise, $p \nvDash_{\mathsf{f}} e^+$). Regarding such bounds for positive examples, we can similarly extend them to negative examples and sets of examples.

In this domain, we guarantee complete calculations just taking $\mathsf{f}(n) = n + 1$, and $p \vDash S$ denotes that p satisfies every example of $S \subset X$ (in the general framework, $p \vDash_{\mathsf{f}} S$

denotes that p satisfies every example of $S \subset X$ within a common maximum value of time steps that depends on f).

Also, it is important to note that even employing large enough complexity functions, language L is *not universal* in the sense that it cannot identify every concept $c \in C$. For instance, given the concept $c = \{\mathsf{NE}^+, \mathsf{W}^+\}$, then $[c]_L = \{\emptyset\}$, i.e., there are no programs in language L that satisfies both NE^+ and W^+. This is because the initial state, q_0, can connect with just one different state q_1. In other words, you cannot choose moving to q_1 or q_2 from state q_0 in just one edge crossing (see Fig. 11).

Appendix B: Examples

In this section we will analyse other examples with more situations and concepts slightly different than in Example 1.

Let us consider another set of concepts where there is just one minimal curriculum: there is no draw between the overall teaching sizes of optimal curricula (as it happens in Example 1).

Example 4 Let us consider the set of concepts $Q' = \{a', b', c\}$, where $\mathsf{RURD} \in [a']_L$, $\mathsf{RRURD} \in [b']_L$ and $\mathsf{R)URD)} \in [c]_L$, are the first programs in their equivalent classes with respect to $\prec$.

Regarding the overall teaching sizes and the descriptions measured in bits (Table 9), Fig. 12 compares some relevant curricula registered in Table 10 for $Q' = \{a', b', c\}$.

As we can see in Table 10 (and Fig. 12), there is only one minimal teaching size curriculum (with no ties). Such minimal curriculum, π'_7, minimises the overall teaching size, placing concept c in a separate branch. If we choose curriculum π'_4, instead of π'_7, we get $\approx 53.3\%$ extra cost.

We observe that for this particular set of concepts Q', if the learner identifies concept c, it should not implement such concept to identify the rest of the concepts for the purpose of minimising the overall teaching size.

As before, we utilise Table 9, containing the teaching sizes associated to Example 4, to build another Table 10 showing the overall teaching size for all the curricula associated.

Finally, we examine another set of concepts, but now with four elements.

Example 5 Let $Q'' = \{a, b, c, d\}$ be the set of concepts where $\mathsf{RURD)} \in [a]_L$, $\mathsf{RRURD)} \in [b]_L$, $\mathsf{R)URD)} \in [c]_L$ and $\mathsf{RD)} \in [d]_L$ are the first programs in their equivalent classes with respect to $\prec$.

Now, there are 73 different curricula (according to Proposition 1) and 64 teaching size values associated to Example 5.

Regarding the overall teaching sizes and the descriptions measured in bits, Fig. 13 compares some relevant curricula registered in Table 12 for Example 5. If we choose π''_{22} or π''_{26}, instead of the optimal curriculum π''_{10}, we get $\approx 66.6\%$ extra cost.

Table 9 Conditional teaching sizes for Example 4

TS (bits)	Teacher-Learner identification
$TS(a') = 15$	$\Phi(\{\text{ENES}^+\}) = \text{RURD}$
$TS(b') = 18$	$\Phi(\{\text{EENES}^+\}) = \text{RRURD}$
$TS(c) = 24$	$\Phi(\{\text{ENESEEE}^+\}) = \text{R)URD)}$
$TS(a'\|b') = 15$	$\Phi(\{\text{ENES}^+\}\|B) = \text{RURD}$
$TS(b'\|a') = 12$	$\Phi(\{\text{EEN}^+\}\|B) = \text{R@}$
$TS(c\|a') = 30$	$\Phi(\{\text{ENESEEEEE}^+\}\|B) = \text{R)URD)}$
$TS(a'\|c) = 24$	$\Phi(\{\text{ENES}^+, \text{EE}^-\}\|B) = \text{RURD}$
$TS(b'\|c) = 30$	$\Phi(\{\text{EENES}^+, \text{EEE}^-\}\|B) = \text{RRURD}$
$TS(c\|b') = 27$	$\Phi(\{\text{ENESEEEE}^+\}\|B) = \text{R)URD)}$
$TS(a'\|b', c) = 24$	$\Phi(\{\text{ENES}^+, \text{EE}^-\}\|B) = \text{RURD}$
$TS(b'\|a', c) = 21$	$\Phi(\{\text{EEN}^+, \text{EN}^-\}\|B) = \text{R@0}$
$TS(c\|a', b') = 33$	$\Phi(\{\text{ENESEEEEEE}^+\}\|B) = \text{R)URD)}$
$TS(a'\|c, b') = 24$	$\Phi(\{\text{ENES}^+, \text{EE}^-\}\|B) = \text{RURD}$
$TS(b'\|c, a') = 24$	$\Phi(\{\text{EEN}^+, \text{EEE}^-\}\|B) = \text{R@1}$
$TS(c\|b', a') = 33$	$\Phi(\{\text{ENESEEEEEE}^+\}\|B) = \text{R)URD)}$

Table 10 Curricula for Example 4

Curriculum	Overall Teaching Size (bits)
$\pi'_0 = \{a', b', c\}$	$TS(\pi'_0) = TS(a') + TS(b') + TS(c) = 15 + 18 + 24 = 57$
$\pi'_1 = \{a' \to b' \to c\}$	$TS(\pi'_1) = TS(a') + TS(b'\|a') + TS(c\|a', b') = 15 + 12 + 33 = 60$
$\pi'_2 = \{b' \to a' \to c\}$	$TS(\pi'_2) = TS(b') + TS(a'\|b') + TS(c\|b', a') = 18 + 15 + 33 = 66$
$\pi'_3 = \{c \to a' \to b'\}$	$TS(\pi'_3) = TS(c) + TS(a'\|c) + TS(b'\|c, a') = 24 + 24 + 24 = 72$
$\pi'_4 = \{c \to b' \to a'\}$	$TS(\pi'_4) = TS(c) + TS(b'\|c) + TS(a'\|c, b') = 24 + 30 + 24 = \mathbf{78}$
$\pi'_5 = \{a' \to c \to b'\}$	$TS(\pi'_5) = TS(a') + TS(c\|a') + TS(b'\|a', c) = 15 + 30 + 21 = 66$
$\pi'_6 = \{b' \to c \to a'\}$	$TS(\pi'_6) = TS(b') + TS(c\|b') + TS(a'\|b', c) = 18 + 27 + 24 = 69$
$\pi'_7 = \{a' \to b', c\}$	$TS(\pi'_7) = TS(a') + TS(b'\|a') + TS(c) = 15 + 12 + 24 = \mathbf{51}$
$\pi'_8 = \{b \to a, c\}$	$TS(\pi'_8) = TS(b') + TS(a'\|b') + TS(c) = 18 + 15 + 24 = 57$
$\pi'_9 = \{a' \to c, b'\}$	$TS(\pi'_9) = TS(a') + TS(c\|a') + TS(b') = 15 + 30 + 18 = 63$
$\pi'_{10} = \{c \to a', b'\}$	$TS(\pi'_{10}) = TS(c) + TS(a'\|c) + TS(b') = 24 + 24 + 18 = 66$
$\pi'_{11} = \{b' \to c, a'\}$	$TS(\pi'_{11}) = TS(b') + TS(c\|b') + TS(a') = 18 + 27 + 15 = 60$
$\pi'_{12} = \{c \to b', a'\}$	$TS(\pi'_{12}) = TS(c) + TS(b'\|c) + TS(a') = 24 + 30 + 15 = 69$

Appendix C Some requirements for our drawing domain

This section aims to identify the conditions for our drawing domain to verify Corollary 2. This section relies in Section A to deal with the notation and the settings of our drawing domain.

Firstly, it is important to know how many commands are necessary to identify a given program. In order to do so, we will consider some kind of paths, as short as possible, on the graph defined by the DFA corresponding to a given program. Those paths would be denoted as sequences of commands that represent the successive edges included. For

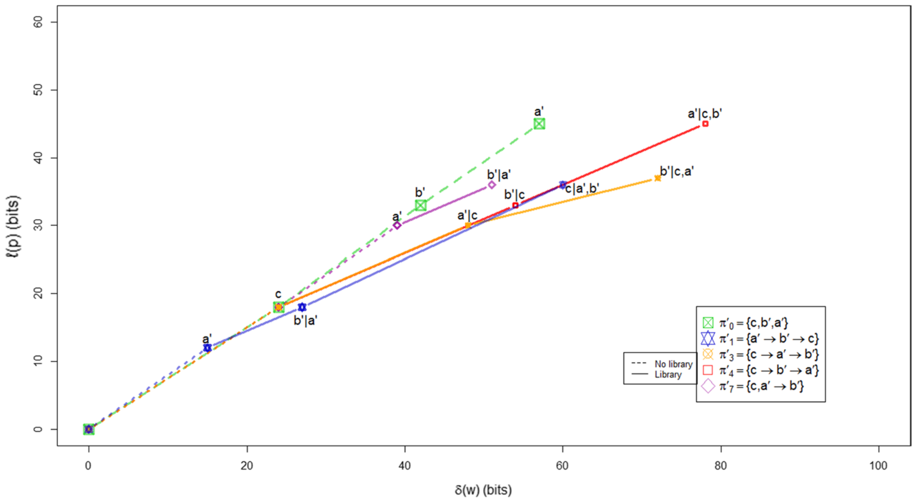

Fig. 12 Some relevant curricula comparison in Example 4 (Table 10)

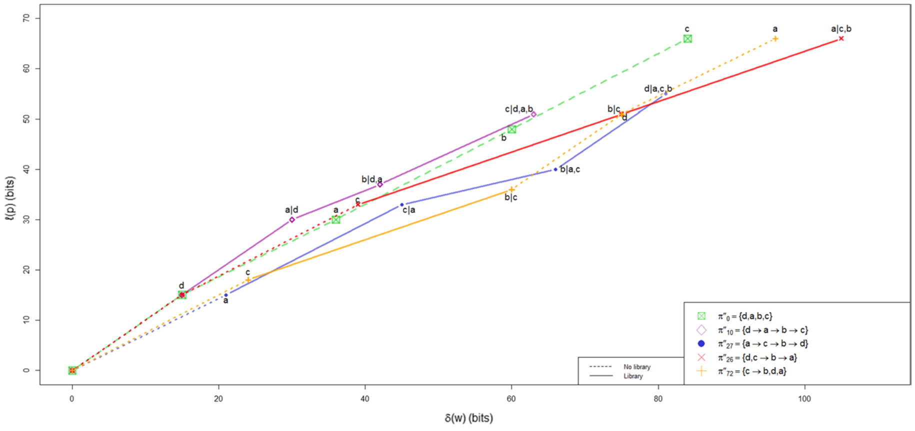

Fig. 13 Some relevant curricula comparison in Example 5 (Table 12)

instance, the DFA of Fig. 7, given by the program RU $\in L$, starting at q_0 has just one path that includes all its edges: EN (i.e., $\overrightarrow{q_0q_1}$ and $\overrightarrow{q_1q_2}$ in that order).

If the program has non-equivalent instructions, like RURD), the learner needs not only ENESE, i.e., a path that includes every edge of the DFA, starting at q_0, but also, crossing the edge labelled as ε implies going through $\overrightarrow{q_0q_1}$ too. For instance, if we provide the learner with ENESE$^+$, the learner does not output RURD), but RURDR, because of lexicographical order; that is why we need to add one more command ENESEN$^+$ to make RURD) elegible by the learner.

These kind of paths define a polygonal chain $s \in \Sigma_4^*$ and we denote its *length* as $\dot{\delta}(s)$. We extend such notation to any set $S \subset X$, as $\dot{\delta}(S)$, by considering the number of commands included in S. For instance, $\dot{\delta}(\{\text{EEN}^+, \text{EE}^-\}) = 5$.

Table 11 Teaching sizes for concepts of Example 5 (missing entries are in Table 1)

TS (bits)	Teacher-Learner identification	$TS(b\|d,c) = 30$	$\Phi(\{\mathsf{EENES}^+, \mathsf{EEE}^-\}\|B) = \mathsf{RRU@0}$
$TS(d) = 15$	$\Phi(\{\mathsf{ESES}^+\}) = \mathsf{RD)}$	$TS(d\|a,b,c) = 15$	$\Phi(\{\mathsf{ESES}^+\}\|B) = \mathsf{RD)}$
$TS(a\|d) = 15$	$\Phi(\{\mathsf{ENES}^+\}\|B) = \mathsf{RU@}$	$TS(c\|a,b,d) = 21$	$\Phi(\{\mathsf{ENESEE}^+\}\|B) = \mathsf{R)U@10}$
$TS(b\|d) = 18$	$\Phi(\{\mathsf{EENES}^+\}\|B) = \mathsf{RRU@}$	$TS(d\|a,c,b) = 15$	$\Phi(\{\mathsf{ESES}^+\}\|B) = \mathsf{RD)}$
$TS(c\|d) = 21$	$\Phi(\{\mathsf{ENESEE}^+\}\|B) = \mathsf{R)U@}$	$TS(b\|a,c,d) = 24$	$\Phi(\{\mathsf{EENE}^+, \mathsf{EN}^-\}\|B) = \mathsf{R@00}$
$TS(d\|a) = 15$	$\Phi(\{\mathsf{ESES}^+\}\|B) = \mathsf{RD)}$	$TS(b\|a,d,c) = 24$	$\Phi(\{\mathsf{EENE}^+, \mathsf{EN}^-\}\|B) = \mathsf{R@00}$
$TS(d\|b) = 15$	$\Phi(\{\mathsf{ESES}^+\}\|B) = \mathsf{RD)}$	$TS(d\|b,a,c) = 15$	$\Phi(\{\mathsf{ESES}^+\}\|B) = \mathsf{RD)}$
$TS(d\|c) = 15$	$\Phi(\{\mathsf{ESES}^+\}\|B) = \mathsf{RD)}$	$TS(c\|b,a,d) = 21$	$\Phi(\{\mathsf{ENESEE}^+\}\|B) = \mathsf{R)U@10}$
$TS(d\|a,b) = 15$	$\Phi(\{\mathsf{ESES}^+\}\|B) = \mathsf{RD)}$	$TS(d\|c,a,b) = 15$	$\Phi(\{\mathsf{ESES}^+\}\|B) = \mathsf{RD)}$
$TS(b\|a,d) = 12$	$\Phi(\{\mathsf{EEN}^+\}\|B) = \mathsf{R@0}$	$TS(b\|c,a,d) = 27$	$\Phi(\{\mathsf{EENE}^+, \mathsf{EEE}^-\}\|B) = \mathsf{R@01}$
$TS(d\|b,a) = 15$	$\Phi(\{\mathsf{ESES}^+\}\|B) = \mathsf{RD)}$	$TS(c\|d,a,b) = 21$	$\Phi(\{\mathsf{ENESEE}^+\}\|B) = \mathsf{R)U@00}$
$TS(a\|b,d) = 15$	$\Phi(\{\mathsf{ENES}^+\}\|B) = \mathsf{RU@1}$	$TS(b\|d,a,c) = 24$	$\Phi(\{\mathsf{EENE}^+, \mathsf{EN}^-\}\|B) = \mathsf{R@01}$
$TS(b\|d,a) = 15$	$\Phi(\{\mathsf{EEN}^+\}\|B) = \mathsf{R@1}$	$TS(d\|b,c,a) = 15$	$\Phi(\{\mathsf{ESES}^+\}\|B) = \mathsf{RD)}$
$TS(a\|d,b) = 15$	$\Phi(\{\mathsf{ENES}^+\}\|B) = \mathsf{RU@0}$	$TS(c\|b,d,a) = 21$	$\Phi(\{\mathsf{ENESEE}^+\}\|B) = \mathsf{R)U@01}$
$TS(d\|a,c) = 15$	$\Phi(\{\mathsf{ESES}^+\}\|B) = \mathsf{RD)}$	$TS(d\|c,b,a) = 15$	$\Phi(\{\mathsf{ESES}^+\}\|B) = \mathsf{RD)}$
$TS(c\|a,d) = 21$	$\Phi(\{\mathsf{ENESEE}^+\}\|B) = \mathsf{R)U@1}$	$TS(b\|c,d,a) = 27$	$\Phi(\{\mathsf{EENE}^+, \mathsf{EEE}^-\}\|B) = \mathsf{R@10}$
$TS(d\|c,a) = 15$	$\Phi(\{\mathsf{ESES}^+\}\|B) = \mathsf{RD)}$	$TS(c\|d,b,a) = 21$	$\Phi(\{\mathsf{ENESEE}^+\}\|B) = \mathsf{R)U@00}$
$TS(a\|c,d) = 24$	$\Phi(\{\mathsf{ENES}^+, \mathsf{EE}^-\}\|B) = \mathsf{RU@1}$	$TS(b\|d,c,a) = 27$	$\Phi(\{\mathsf{EENE}^+, \mathsf{EEE}^-\}\|B) = \mathsf{R@10}$
$TS(c\|d,a) = 21$	$\Phi(\{\mathsf{ENESEE}^+\}\|B) = \mathsf{R)U@0}$	$TS(a\|b,c,d) = 27$	$\Phi(\{\mathsf{ENESE}^+, \mathsf{EE}^-\}\|B) = \mathsf{RU@10}$
$TS(a\|d,c) = 24$	$\Phi(\{\mathsf{ENES}^+, \mathsf{EE}^-\}\|B) = \mathsf{RU@0}$	$TS(a\|b,d,c) = 27$	$\Phi(\{\mathsf{ENESE}^+, \mathsf{EE}^-\}\|B) = \mathsf{RU@01}$
$TS(d\|b,c) = 15$	$\Phi(\{\mathsf{ESES}^+\}\|B) = \mathsf{RD)}$	$TS(a\|c,b,d) = 27$	$\Phi(\{\mathsf{ENESE}^+, \mathsf{EE}^-\}\|B) = \mathsf{RU@10}$
$TS(c\|b,d) = 21$	$\Phi(\{\mathsf{ENESEE}^+\}\|B) = \mathsf{R)U@1}$	$TS(a\|c,d,b) = 27$	$\Phi(\{\mathsf{ENESE}^+, \mathsf{EE}^-\}\|B) = \mathsf{RU@01}$
$TS(d\|c,b) = 15$	$\Phi(\{\mathsf{ESES}^+\}\|B) = \mathsf{RD)}$	$TS(a\|d,b,c) = 27$	$\Phi(\{\mathsf{ENESE}^+, \mathsf{EE}^-\}\|B) = \mathsf{RU@00}$
$TS(b\|c,d) = 30$	$\Phi(\{\mathsf{EENES}^+, \mathsf{EEE}^-\}\|B) = \mathsf{RRU@1}$	$TS(a\|d,c,b) = 27$	$\Phi(\{\mathsf{ENESE}^+, \mathsf{EE}^-\}\|B) = \mathsf{RU@00}$
$TS(c\|d,b) = 21$	$\Phi(\{\mathsf{ENESEE}^+\}\|B) = \mathsf{R)U@0}$	$TS(c\|a,d,b) = 21$	$\Phi(\{\mathsf{ENESEE}^+\}\|B) = \mathsf{R)U@01}$

Note that every time we *go through* an ε-edge of the DFA, it is compulsory to subsequently include the edge starting at q_0 labelled with a command. As a result, the learner needs at least seven commands to identify program R)URD) (Fig. 10). In short, $\Phi(w) = \mathsf{R)URD)}$ implies $\dot{\delta}(w) \geq 7$; in fact, $\Phi(\{\mathsf{ENESEEE}^+\}) = \mathsf{R)URD)}$ (see Table 1).

There are some programs that show even more differences between its number of instructions and the length of examples accepted. For instance, the program UDRL)DLUR defines the DFA of Fig. 14 with nine instructions, but the learner needs at least a sequence of thirteen commands labelled as a positive example to identify such program.

Apart from that, it is useful to know the number of instructions of a program when its library calls are *unfolded*, i.e., given a program p and a library B, we use $\circ(p)$ to denote the program that is equivalent to p, where each primitive call @ has been replaced by the instructions of the primitive in B. We suppose that $\circ(p)$ works exactly equal in L as p in L_B, i.e., the language L using library B.

Let us suppose now that we use library $B = \langle \mathsf{DLUR} \rangle$. In such particular case, UDRL)DLUR in L is expressed as UDRL)@ in L_B, by removing $4 - 1 = 3$ instructions, i.e., $\dot{\ell}(\mathsf{UDRL)DLUR}) = \dot{\ell}(\circ(\mathsf{UDRL)@}))+3$.

With respect to the minimum length of the paths associated to UDRL)@, in L_B we reduce such minimum length in $4 - 2 = 2$ units; so that the learner needs at least a sequence of

Table 12 Curricula for Example 5

Curriculum	Overall bits	$\pi_{36} = \{b \to c \to a, d\}$	$TS(\pi''_{36}) = 93$
$\pi''_{0} = \{a, b, c, d\}$	$TS(\pi''_{0}) = 84$	$\pi''_{37} = \{a \to b \to d, c\}$	$TS(\pi''_{37}) = 72$
$\pi''_{1} = \{a \to d, b, c\}$	$TS(\pi''_{1}) = 84$	$\pi''_{38} = \{a \to d \to b, c\}$	$TS(\pi''_{38}) = 72$
$\pi''_{2} = \{a, b \to d, c\}$	$TS(\pi''_{2}) = 84$	$\pi''_{39} = \{d \to a \to b, c\}$	$TS(\pi''_{39}) = 66$
$\pi''_{3} = \{a, b, c \to d\}$	$TS(\pi''_{3}) = 84$	$\pi''_{40} = \{a \to b, c \to d\}$	$TS(\pi_{40}) = 72$
$\pi''_{4} = \{d \to a, b, c\}$	$TS(\pi''_{4}) = 78$	$\pi''_{41} = \{a \to b, d \to c\}$	$TS(\pi''_{41}) = 69$
$\pi''_{5} = \{a, d \to b, c\}$	$TS(\pi''_{5}) = 78$	$\pi''_{42} = \{a \to b, c, d\}$	$TS(\pi''_{42}) = 72$
$\pi''_{6} = \{a, b, d \to c\}$	$TS(\pi''_{6}) = 81$	$\pi''_{43} = \{b \to a \to d, c\}$	$TS(\pi''_{43}) = 84$
$\pi''_{7} = \{a \to b \to c \to d\}$	$TS(\pi''_{7}) = 72$	$\pi''_{44} = \{b \to d \to a, c\}$	$TS(\pi''_{44}) = 78$
$\pi''_{8} = \{a \to b \to d \to c\}$	$TS(\pi''_{8}) = 69$	$\pi''_{45} = \{d \to b \to a, c\}$	$TS(\pi''_{45}) = 72$
$\pi''_{9} = \{a \to d \to b \to c\}$	$TS(\pi''_{9}) = 69$	$\pi''_{46} = \{b \to a, c \to d\}$	$TS(\pi''_{46}) = 84$
$\pi''_{10} = \{d \to a \to b \to c\}$	$TS(\pi''_{10}) = \mathbf{63}$	$\pi''_{47} = \{b \to a, d \to c\}$	$TS(\pi''_{47}) = 81$
$\pi''_{11} = \{a \to b \to c, d\}$	$TS(\pi''_{11}) = 72$	$\pi''_{48} = \{b \to a, c, d\}$	$TS(\pi''_{48}) = 84$
$\pi''_{12} = \{b \to a \to c \to d\}$	$TS(\pi''_{12}) = 84$	$\pi''_{49} = \{a \to c \to d, b\}$	$TS(\pi''_{49}) = 84$
$\pi''_{13} = \{b \to a \to d \to c\}$	$TS(\pi''_{13}) = 81$	$\pi''_{50} = \{a \to d \to c, b\}$	$TS(\pi''_{50}) = 81$
$\pi''_{14} = \{b \to d \to a \to c\}$	$TS(\pi''_{14}) = 75$	$\pi''_{51} = \{d \to a \to c, b\}$	$TS(\pi''_{51}) = 75$
$\pi''_{15} = \{d \to b \to a \to c\}$	$TS(\pi''_{15}) = 69$	$\pi''_{52} = \{a \to c, b \to d\}$	$TS(\pi''_{52}) = 84$
$\pi''_{16} = \{b \to a \to c, d\}$	$TS(\pi''_{16}) = 84$	$\pi''_{53} = \{a \to c, d \to b\}$	$TS(\pi''_{53}) = 78$
$\pi''_{17} = \{c \to a \to b \to d\}$	$TS(\pi''_{17}) = 93$	$\pi''_{54} = \{a \to c, b, d\}$	$TS(\pi''_{54}) = 84$
$\pi''_{18} = \{c \to a \to d \to b\}$	$TS(\pi''_{18}) = 96$	$\pi''_{55} = \{c \to a \to d, b\}$	$TS(\pi''_{55}) = 93$
$\pi''_{19} = \{c \to d \to a \to b\}$	$TS(\pi''_{19}) = 90$	$\pi_{56} = \{c \to d \to a, b\}$	$TS(\pi''_{56}) = 87$
$\pi''_{20} = \{d \to c \to a \to b\}$	$TS(\pi''_{20}) = 87$	$\pi''_{57} = \{d \to c \to a, b\}$	$TS(\pi''_{57}) = 84$
$\pi''_{21} = \{c \to a \to b, d\}$	$TS(\pi''_{21}) = 93$	$\pi''_{58} = \{c \to a, b \to d\}$	$TS(\pi''_{58}) = 93$
$\pi''_{22} = \{c \to b \to a \to d\}$	$TS(\pi''_{22}) = \mathbf{105}$	$\pi''_{59} = \{c \to a, d \to b\}$	$TS(\pi''_{59}) = 87$
$\pi''_{23} = \{c \to b \to d \to a\}$	$TS(\pi''_{23}) = 102$	$\pi''_{60} = \{c \to a, b \to d\}$	$TS(\pi''_{60}) = 93$
$\pi''_{24} = \{c \to d \to b \to a\}$	$TS(\pi''_{24}) = 96$	$\pi''_{61} = \{b \to c \to d, a\}$	$TS(\pi''_{61}) = 84$
$\pi''_{25} = \{d \to c \to b \to a\}$	$TS(\pi''_{25}) = 93$	$\pi''_{62} = \{b \to d \to c, a\}$	$TS(\pi''_{62}) = 81$
$\pi''_{26} = \{c \to b \to a, d\}$	$TS(\pi''_{26}) = \mathbf{105}$	$\pi''_{63} = \{d \to b \to c, a\}$	$TS(\pi_{63}) = 75$
$\pi''_{27} = \{a \to c \to b \to d\}$	$TS(\pi''_{27}) = 81$	$\pi''_{64} = \{b \to c, a \to d\}$	$TS(\pi''_{64}) = 84$
$\pi''_{28} = \{a \to c \to d \to b\}$	$TS(\pi''_{28}) = 84$	$\pi''_{65} = \{b \to c, d \to a\}$	$TS(\pi''_{65}) = 78$
$\pi''_{29} = \{a \to d \to c \to b\}$	$TS(\pi''_{12}) = 84$	$\pi''_{66} = \{b \to c, a \to d\}$	$TS(\pi''_{66}) = 84$
$\pi''_{30} = \{d \to a \to c \to b\}$	$TS(\pi''_{30}) = 75$	$\pi''_{67} = \{c \to b \to d, a\}$	$TS(\pi''_{67}) = 96$
$\pi''_{31} = \{a \to c \to b, d\}$	$TS(\pi''_{31}) = 81$	$\pi''_{68} = \{c \to d \to b, a\}$	$TS(\pi''_{68}) = 90$
$\pi''_{32} = \{b \to c \to a \to d\}$	$TS(\pi''_{32}) = 93$	$\pi''_{69} = \{d \to c \to b, a\}$	$TS(\pi''_{69}) = 87$
$\pi''_{33} = \{b \to c \to d \to a\}$	$TS(\pi''_{33}) = 90$	$\pi''_{70} = \{c \to b, a \to d\}$	$TS(\pi''_{70}) = 96$
$\pi''_{34} = \{b \to d \to c \to a\}$	$TS(\pi''_{34}) = 87$	$\pi''_{71} = \{c \to b, d \to a\}$	$TS(\pi''_{71}) = 90$
$\pi''_{35} = \{d \to b \to c \to a\}$	$TS(\pi''_{35}) = 81$	$\pi''_{72} = \{c \to b, a, d\}$	$TS(\pi''_{72}) = 96$

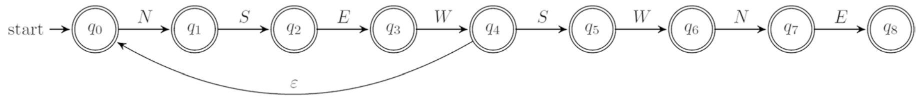

Fig. 14 Automaton identified by the expression UDRL)DLUR

eleven succesive commands to identify UDRL)@. This is because when a library call is the last instruction, it only needs to be *fed* twice, i.e., it needs two commands. For instance, in our case we just feed instruction @ in UDRL)@ with SW, i.e., two commands corresponding to the first two instructions of primitive DLUR.

It can happen that the primitive referred by @, as a final instruction, might include non-equivalent instructions ()-instructions). For instance, if @ points to D)LUR, we could skip instruction), if possible; that is to say, if there are sufficient equivalent instructions in the primitive. For example, we should *feed* @ with SW, corresponding to the 1st and 3rd instructions of primitive D)LUR. Of course, if the primitive were to be D), we could not skip the 2nd instruction.

In the event that the library call is not the last instruction, we just need to feed all the equivalent instructions of the primitive. For instance, if we employ $B = \langle$UDRL)$\rangle$ as library, UDRL)DLUR in L is expressed as @DLUR in L_B, whose shortest path, elegible by the learner is at least nine units length.

We do not consider programs with consecutive ocurrences of)-instructions, like)) or))). Then a program with n instructions, without library calls, has $\lfloor \frac{n}{2} \rfloor$ equivalent instructions minimum and $\lceil \frac{n}{2} \rceil$ maximum.

Once that we have shown some particular examples, we are able to give a definition of minimum eligible path.

We will consider programs without library calls in this definition. If we get a program p with library calls, we will consider $\circ(p)$ instead but with the following exception: if the last instruction of p is a library call, we will only consider two instructions of the primitive referred by strict order of appearance and skipping instructions), if the length of the primitive facilitates it.

Definition 12 Let p be a program in language L without library calls. We say that a *path* on the DFA corresponding to p is *minimum eligible* when satisfies the following properties: (1) The path goes through every edge of the graph, (2) If the path crosses an edge labelled as ε, it is subsequently followed by the edge $\overrightarrow{q_0q_1}$ and 3) If there is any edge labelled as ε, then any path according to previous conditions, 1) and 2), increases its length one more unit.

We now aim to identify sufficient conditions that, in language L, guarantee Inequality (2).

We consider language L_B where $0 \leq |B|$; @i denote library calls when $|B| > 1$ (@ if $|B| = 1$). We hold that the value of the instructions of language L, except instructions @i, have the same value in bits as commands of Σ_4.

We also take the extension for $L_{B'}$ with $B' = \langle B, p \rangle$, where p is a primitive without library calls. Though it is not true in general, we assume that the size in bits of a library call, @i in L_B is equal to a call, @i′ in $L_{B'}$.

Theorem 3 *Let $c \in C_L$ and B, B' libraries such that $\ell(@\mathsf{i}) = \ell(@\mathsf{i}')$, where @i and @i′ are in language L_B and $L_{B'}$, respectively. Let $w_c, w'_c \subset X$ be witness sets with $p^*_{c|B} = \Phi_\ell(w_c|B)$ and $p^*_{c|B'} = \Phi_\ell(w'_c|B')$. If w_c has no negative examples, then*:

$$\delta(w_c) - \delta(w'_c) \leq \ell(p^*_{c|B}) - \ell(p^*_{c|B'}) \tag{C1}$$

Proof Since $c \in C_L$, we can take a large enough complexity function f. Also, we consider that there is just one positive example in w_c, since the learner is always given the shortest witness set. Moreover, such positive example defines a minimum eligible path, otherwise the learner would not be given the simplest witness set. In other words, we can assume the following statements:

1. $\exists w_c, w'_c \subset X$ such that $p^*_{c|B} = \Phi_\ell(w_c|B)$, $p^*_{c|B'} = \Phi_\ell(w'_c|B')$
2. The witness set is $w_c = \{e^+\}$, and e^+ is a minimum eligible path on the DFA defined by $p^*_{c|B}$

The number of steps in such minimum path defined by e^+ is given by $\dot{\delta}(w_c)$, while the number of instructions of $p^*_{c|B}$ is $\dot{\ell}(p^*_{c|B})$. Then,

$$\dot{\delta}(w_c) = \dot{\ell}(p^*_{c|B}) + n, \text{ with } n \in \mathbb{N}(n \geq 0) \tag{C2}$$

Let us suppose that $\dot{\ell}(p^*_{c|B'}) = \dot{\ell}(p^*_{c|B})$, i.e., language $L_{B'}$ does not reduce the number of instructions of $p^*_{c|B}$, then $p^*_{c|B'} = p^*_{c|B}$. So that, $w'_c = w_c$, and the conclusion (C1) is guaranteed.

Let us suppose that $\dot{\ell}(p^*_{c|B'}) < \dot{\ell}(p^*_{c|B})$. Worst-case scenario implies just one new library call, denoted by $@\mathsf{i}'$, to reduce $p^*_{c|B}$ to $p^*_{c|B'}$. So that, $\dot{\ell}(p^*_{c|B'})$ decreases $\dot{\ell}(p^*_{c|B})$ in $\dot{\ell}(\circ(@\mathsf{i}')) - 1$ units. We observe that, if there were more library calls $@\mathsf{i}'$ in $p^*_{c|B'}$, the reduction would be larger. Hence,

$$\dot{\ell}(p^*_{c|B'}) = \dot{\ell}(p^*_{c|B}) - (\dot{\ell}(\circ(@\mathsf{i}')) - 1) \tag{C3}$$

We now distinguish three sub-cases according to its position and the number of instructions of the new primitive, $@\mathsf{i}'$, included in B'.

Case 1: We suppose that $\dot{\delta}(\circ(@\mathsf{i}')) > 2$ and $@\mathsf{i}'$ is the last instruction in $p^*_{c|B'}$. In this case, we can feed $@\mathsf{i}'$ with just two commands.

Hence, considering the library call, the path expressed by the positive example included in w'_c will verify at the very least:

$$\dot{\delta}(w'_c) = \dot{\delta}(w_c) - (\dot{\ell}(\circ(@\mathsf{i}')) - 2) \tag{C4}$$

Substituting (C3) and (C4) in (C2) we obtain

$$\dot{\delta}(w'_c) + \dot{\ell}(\circ(@\mathsf{i}')) - 2 = \dot{\ell}(p^*_{c|B'}) + \dot{\ell}(\circ(@\mathsf{i}')) - 1 + n, \text{ with } n \in \mathbb{N}$$

Therefore:

$$\dot{\delta}(w'_c) = \dot{\ell}(p^*_{c|B'}) + (n + 1) \tag{C5}$$

Using equalities (C2) and (C5), we get

$$\dot{\delta}(w_c) - \dot{\delta}(w'_c) = \dot{\ell}(p^*_{c|B}) - \dot{\ell}(p^*_{c|B'}) - 1 \tag{C6}$$

Now, we consider that $p^*_{c|B'}$ has the same number of library calls $@\mathsf{i}$ (except $@\mathsf{i}'$). Otherwise, the difference $\ell(p^*_{c|B}) - \ell(p^*_{c|B'})$ would be even greater, since the learner always retrieves the shortest option.

So that, we can express in bits (C6) as the following inequality

$$\delta(w_c) - \delta(w'_c) \leq \ell(p^*_{c|B}) - \ell(p^*_{c|B'}),$$

which is the conclusion (C1).

Case 2: Now, we suppose that $\dot{\delta}(\circ(@\mathsf{i}')) = 2$ and $@\mathsf{i}'$ is the last instruction of $p^*_{c|B'}$. Then, we need to feed $@\mathsf{i}'$ with two commands: one to feed the call and another to avoid the *substitution* of such call by an instruction equivalent to a command; otherwise, the learner would have chosen a program previous to $p^*_{c|B'}$, which is not possible.

Hence, $\dot{\delta}(w'_c) = \dot{\delta}(w_c)$, i.e., we can not reduce the teaching size and we get that $\delta(w_c) - \delta(w'_c) = 0$. Since $\ell(p^*_{c|B}) - \ell(p^*_{c|B'}) \geq 0$, then we get the conclusion (C1).

Case 3: In this case, the library call $@\mathsf{i}'$ is not the last instruction. At most, we could skip the)-instructions included in $\circ(@\mathsf{i}')$, but we should feed its equivalent instructions. So that, at the very least, there must be $m = \lfloor \frac{\dot{\ell}(\circ(@\mathsf{i}'))}{2} \rfloor$ instructions equivalent to commands in $@\mathsf{i}'$, with $m \geq 1$.

Therefore, at the very least we have $\dot{\delta}(w'_c) = \dot{\delta}(w_c) - (\dot{\ell}(\circ(@\mathsf{i}')) - m)$.

Also, we can remove $\dot{\ell}(p^*_{c|B'}) = \dot{\ell}(p^*_{c|B}) - (\dot{\ell}(\circ(@\mathsf{i}') - 1))$. Using (C2) we get that

$$\dot{\delta}(w'_c) + \dot{\ell}(\circ(@\mathsf{i}') - m = \dot{\ell}(p^*_{c|B'}) + \dot{\ell}(\circ(@\mathsf{i}') - 1 + n$$

Then, we obtain $\dot{\delta}(w'_c) = \dot{\ell}(p^*_{c|B'}) + n + (m-1)$, which together with (C2) provides:

$$\dot{\delta}(w_c) - \dot{\delta}(w'_c) = \dot{\ell}(p^*_{c|B}) - \dot{\ell}(p^*_{c|B'}) - (m-1) \tag{C7}$$

Since $m \geq 1$ and making a translation into bits, similar to the one already done in Case 1, we get the conclusion (C1). □

Appendix D: Some proofs

In this section we will include the proofs of two theoretical statements. The following result is useful to prove Corollary 2.

Lemma 4 *Let $x, x', y, y' \in \mathbb{N}^*$ verify the following conditions:*

1. $\exists k_1, k_2 \in \mathbb{N}$ *such that* $x = y + k_1$ *and* $x' = y' + k_2$
2. $x - x' \leq y - y'$
3. $y' \leq y$

Then, $\frac{y'}{y} \leq \frac{x'}{x}$.

Proof Using Condition 1, we know that

$$x + x' = y + k_1 + y' + k_2 \tag{D8}$$

Condition 2 provides that $\exists \epsilon \geq 0$ such that

$$x - x' + \epsilon = y - y' \tag{D9}$$

From (D8) and (D9), we get that

$$2x + \epsilon = 2y + k_1 + k_2 \tag{D10}$$

Now, the first part of Condition 1 assures that

$$2x = 2y + 2k_1 \tag{D11}$$

If we substitute (D11) in (D10), we get that

$$2y + 2k_1 + \epsilon = 2y + k_1 + k_2$$

In other words: $\exists \epsilon \geq 0$ such that $k_2 = k_1 + \epsilon$. Therefore:

$$k_1 \leq k_2 \tag{D12}$$

Utilising Condition 3 we observe that

$$\frac{y'}{y} \leq \frac{y' + k_2}{y + k_2} = \frac{x'}{y + k_2} \tag{D13}$$

Also, since $k_2 \geq k_1$ (D12) we can state that

$$\frac{x'}{y + k_2} \leq \frac{x'}{y + k_1} = \frac{x'}{x} \tag{D14}$$

As a consequence of (D13) and (D14) we get $\frac{y'}{y} \leq \frac{x'}{x}$, which completes the proof. □

We now provide the proof of Corollary 2 (Sect. 5).

Proof Lemma 4 can be applied to those situations where $x = \delta(w_c)$, $x' = \delta(w'_c)$, $y = \ell(p_c)$ and $y' = \ell(p'_c)$. In such state, it provides a sufficient condition to obtain a lower bound of the unknown conditional teaching size using a new primitive. Thus, $\frac{\ell(p'_c)}{\ell(p_c)} \leq \frac{\delta(w'_c)}{\delta(w_c)}$ implies the conclusion of Corollary 2. □

Author Contributions MGP and JHO conceived the main ideas of the paper and wrote the paper. The theoretical and experimental results were performed by MGP.

Funding Open Access funding provided thanks to the CRUE-CSIC agreement with Springer Nature. This work was funded by the Norwegian Research Council grant 329745 Machine Teaching for Explainable AI, the EU (FEDER) and Spanish grant RTI2018-094403-B-C32 funded by MCIN/AEI/10.13039/501100011033 and by "ERDF A way of making Europe", Generalitat Valenciana under PROMETEO/2019/098, EU's Horizon 2020 research and innovation programme under grant agreement No. 952215 (TAILOR) and and Spanish grant PID2021-122830OB-C42 (SFERA) funded by MCIN/AEI/10.13039/501100011033 and ERDF A way of making Europe.

Availability of data and materials All data of the experiments are synthetic and they can be found at https://github.com/mgpiqueras/curricula_data.git.

Code availability The software was developed specifically for this study and it can be found at https://github.com/mgpiqueras/curricula_code.git

Declarations

Conflict of interest None.

Ethical approval N/A.

Consent to participate N/A.

Consent for publication Everything included in this paper has been produced by the authors and both give their consent for publication.

References

Barbehenn, M. (1998). A note on the complexity of Dijkstra's algorithm for graphs with weighted vertices. *IEEE Transactions on Computers, 47*(2), 263. https://doi.org/10.1109/12.663776

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

Clayton, N.R., & Abbass, H. (2019). Machine teaching in hierarchical genetic reinforcement learning: Curriculum design of reward functions for swarm shepherding. In *2019 IEEE congress on evolutionary computation (CEC)*, pp. 1259–1266 . https://doi.org/10.1109/CEC.2019.8790157.

Das, R., Tenenbaum, J. B., Solar-Lezama, A., & Tavares, Z. (2023). Combining Functional and Automata Synthesis to Discover Causal Reactive Programs. In *Proceedings of the ACM on Programming Languages*, *7*(56), 1–31. https://doi.org/10.1145/3571249

Degen, J., Hawkins, R. D., Graf, C., Kreiss, E., & Goodman, N. D. (2020). When redundancy is useful: A Bayesian approach to "overinformative'' referring expressions. *Psychological Review, 127*(4), 591.

Dijkstra, E. W., et al. (1959). A note on two problems in connexion with graphs. *Numerische mathematik, 1*(1), 269–271.

Dong, T., He, J., Wang, S., Wang, L., Cheng, Y., & Zhong, Y. (2016). Inability to activate Rac1-dependent forgetting contributes to behavioral inflexibility in mutants of multiple autism-risk genes. *Proceedings of the National Academy of Sciences, 113*(27), 7644–7649. https://doi.org/10.1073/pnas.1602152113

Epp, J. R., Mera, R. S., Köhler, S., Josselyn, S. A., & Frankland, P. W. (2016). Neurogenesis-mediated forgetting minimizes proactive interference. *Nature Communications, 7*, 10838. https://doi.org/10.1038/ncomms10838

Forestier, S., Portelas, R., Mollard, Y., & Oudeyer, P.-Y. (2022). Intrinsically motivated goal exploration processes with automatic curriculum learning. *Journal of Machine Learning Research, 23*(152), 1–41.

Garcia-Piqueras, M., & Hernández-Orallo, J. (2021). Optimal teaching curricula with compositional simplicity priors. In *Joint european conference on machine learning and knowledge discovery in databases*, pp. 1–16 . Springer.

Hansen, E. A., & Zhou, R. (2007). Anytime heuristic search. *Journal of Artificial Intelligence Research, 28*, 267–297.

Hart, P. E., Nilsson, N. J., & Raphael, B. (1968). A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics, 4*(2), 100–107.

Lake, B., Salakhutdinov, R., & Tenenbaum, J. (2015). Human-level concept learning through probabilistic program induction. *Science, 350*(6266), 1332–1338. https://doi.org/10.1126/science.aab3050

Marcus, G. (2018) Deep learning: A critical appraisal. https://doi.org/10.48550/ARXIV.1801.00631

McCulloch, W. S., & Pitts, W. (1943). A logical calculus of the ideas immanent in nervous activity. *The Bulletin of Mathematical Biophysics, 5*(4), 115–133.

Pearl, J. (1984). *Heuristics: Intelligent search strategies for computer problem solving*. Boston: Addison-Wesley Longman Publishing Co., Inc.

Pentina, A., Sharmanska, V., & Lampert, C.H. (2015). Curriculum learning of multiple tasks. In *Proceedings of the IEEE conference on computer vision and pattern recognition (CVPR).*

Richards, B. A., & Frankland, P. W. (2017). The persistence and transience of memory. *Neuron, 94*(6), 1071–1084. https://doi.org/10.1016/j.neuron.2017.04.037

Rios, L. H. O., & Chaimowicz, L. (2010). A survey and classification of A* based best-first heuristic search algorithms. In A. C. da Rocha Costa, R. M. Vicari, & F. Tonidandel (Eds.), *Advances in artificial intelligence - SBIA 2010* (pp. 253–262). Berlin: Springer.

Russell, S. J., & Norvig, P. (2020). *Artificial intelligence: A modern approach*. Hoboken: Pearson.

Shrestha, A., & Mahmood, A. (2019). Review of deep learning algorithms and architectures. *IEEE Access, 7*, 53040–53065. https://doi.org/10.1109/ACCESS.2019.2912200

Shuai, Y., Lu, B., Hu, Y., Wang, L., Sun, K., & Zhong, Y. (2010). Forgetting is regulated through Rac activity in drosophila. *Cell, 140*(4), 579–589. https://doi.org/10.1016/j.cell.2009.12.044

Soviany, P., Ionescu, R. T., Rota, P., & Sebe, N. (2022). Curriculum learning: A survey. *International Journal of Computer Vision, 130*(6), 1526–1565. https://doi.org/10.1007/s11263-022-01611-x

Tang, Y., Wang, X., Harrison, A.P., Lu, L., Xiao, J., & Summers, R.M. (2018). Attention-guided curriculum learning for weakly supervised classification and localization of thoracic diseases on chest radiographs. In *International workshop on machine learning in medical imaging*, pp. 249–258. Springer.

Telle, J. A., Hernández-Orallo, J., & Ferri, C. (2019). The teaching size: Computable teachers and learners for universal languages. *Machine Learning, 108*(8), 1653–1675.

Tenenbaum, J. B., De Silva, V., & Langford, J. C. (2000). A global geometric framework for nonlinear dimensionality reduction. *Science, 290*(5500), 2319–2323.

Wang, W., Caswell, I., & Chelba, C. (2019). Dynamically composing domain-data selection with clean-data selection by "co-curricular learning" for neural machine translation. In *Proceedings of the 57th annual meeting of the association for computational linguistics. Association for Computational Linguistics, Florence, Italy*, , pp. 1282–1292. https://doi.org/10.18653/v1/P19-1123.

Wang, J., Wang, X., & Liu, W. (2018). Weakly-and semi-supervised faster r-CNN with curriculum learning. In *2018 24th international conference on pattern recognition (ICPR)*, pp. 2416–2421. IEEE.

Wang, X., Chen, Y., & Zhu, W. (2021). A survey on curriculum learning. *IEEE Transactions on Pattern Analysis and Machine Intelligence*. https://doi.org/10.1109/TPAMI.2021.3069908

Wong, C., Friedman, Y., Andreas, J., & Tenenbaum, J. (2021). Language as a bootstrap for compositional visual reasoning. In *Proceedings of the annual meeting of the cognitive science society*, Vol. 43.

Wu, J., Zhang, C., Xue, T., Freeman, W.T., & Tenenbaum, J.B. (2016). Learning a probabilistic latent space of object shapes via 3d generative-adversarial modeling. In *Proceedings of the 30th international conference on neural information processing systems*, pp. 82–90.

Zhou, T., & Bilmes, J. (2018). Minimax curriculum learning: Machine teaching with desirable difficulties and scheduled diversity. In *International conference on learning representations*.

Zhu, X. (2015). Machine teaching: An inverse problem to machine learning and an approach toward optimal education. In *29th AAAI conference on artificial intelligence*, pp. 4083–4087.

Zhu, X., Singla, A., Zilles, S., & Rafferty, A. (2018). An overview of machine teaching. arXiv:1801.05927.