Consistent Identification in the Limit of Some of Penn and Buszkowski's Classes is NP-hard

Christophe Costa Florêncio[*]
UiL/OTS, Utrecht University

Abstract

In (Buszkowski, 1987) and (Buszkowski and Penn, 1990) certain 'discovery procedures' for classical categorial grammars were defined. These procedures accept a sequence of structures (strings labeled with derivational information) as input and yield a set of hypotheses in the form of grammars.

In (Kanazawa, 1998) learning functions based on these discovery procedures were studied, and it was shown that some of the classes associated with these procedures can be effectively identified in the limit from positive data. The time complexity of these functions however was still left an open question.

In this paper I will show that learning functions for these classes that are responsive and consistent on their class and learn their class prudently are all NP-hard.

1 Identification in the Limit

In the seminal paper (Gold, 1967) the concept of identification in the limit was introduced. In this model of learning a learning function receives an endless stream of sentences from the target language, called a *text*, and hypothesizes a grammar for the target language at each time-step.

A *class* of languages is called *learnable* if and only if there exists a learning function such that after a finite number of presented sentences it guesses the right language on every text for every language from that class and does not deviate from this hypothesis. Research within this framework is known as *formal learnability theory*.

In this paper only those aspects of formal learnability theory that are relevant to the proof of NP-hardness will be discussed. See (Osherson et al., 1997) and (Jain et al., 1999) for a comprehensive overview of the field.

In formal learnability theory the set Ω denotes the hypothesis space, which can be any class of finitary objects. Members of Ω are called *grammars*.

The set $\mathbf{S}$ denotes the sample space, a recursive subset of Σ^* for some fixed finite alphabet Σ. Elements of $\mathbf{S}$ are called *sentences*, subsets of $\mathbf{S}$ (which obviously are sets of sentences) are called *languages*.

The function L maps elements of Ω to subsets of $\mathbf{S}$. If G is a grammar in Ω, then $\mathrm{L}(G)$ is called the *language generated by (associated with) G*. L is also called the *naming function*. The question whether a sentence belongs to a language generated by a grammar is called the *universal membership problem*. Usually, the naming function is assumed to be such that the universal membership problem is decidable or at least semi-decidable (r.e.).[1]

A triple $\langle \Omega, \mathbf{S}, \mathrm{L} \rangle$ satisfying the above conditions is called a *grammar system*. A class of grammars is denoted $\mathcal{G}$, a class of languages is denoted $\mathcal{L}$.

I will adopt notation from (Kanazawa, 1998) and let $\mathcal{FL}$ denote a class of *structure languages*, to be defined in Section 3. The corresponding naming function is $\mathrm{FL}(G)$. Learning functions are written as φ, their input sequences as σ or τ.

[*] I would like to thank Dick de Jongh and Peter van Emde-Boas for their valuable comments.

[1]In fact, in this paper learning is assumed to take place under conditions such that membership is decidable and the class of grammars to be learned is r.e. These are quite natural conditions for linguistically plausible grammar formalisms, and very convenient when dealing with learnability issues. See (Angluin, 1980) and (Wright, 1989) for characterizations of learnable classes under these restrictions.

1.1 Constraints on Learning Functions

The behaviour of learning functions can be constrained in a number of ways. Such a constraint is called *restrictive* if it restricts the space of learnable classes. Only some important constraints relevant to this discussion will be defined here:

Definition 1 ***Consistent Learning***
A learning function φ is consistent *on $\mathcal{G}$ if for any $L \in \mathrm{L}(\mathcal{G})$ and for any finite sequence $\langle s_0, \ldots, s_i \rangle$ of elements of L, either $\varphi(\langle s_0, \ldots, s_i \rangle)$ is undefined or $\{s_0, \ldots, s_i\} \subseteq \mathrm{L}(\varphi(\langle s_0, \ldots, s_i \rangle))$.*

Informally, consistency requires that the learning function explains all the data it sees with its conjecture.

Definition 2 ***Prudent Learning***
A learning function φ learns $\mathcal{G}$ prudently *if φ learns $\mathcal{G}$ and* $\mathrm{range}(\varphi) \subseteq \mathcal{G}$.

Prudent learners only hypothesize grammars that are in the class they are able to learn.

Definition 3 ***Responsive Learning***
A learning function φ is responsive *on $\mathcal{G}$ if for any $L \in \mathrm{L}(\mathcal{G})$ and for any finite sequence $\langle s_0, \ldots, s_i \rangle$ of elements of L ($\{s_0, \ldots, s_i\} \subseteq L$), $\varphi(\langle s_0, \ldots, s_i \rangle)$ is defined.*

A responsive learning function is always defined, as long as the text is consistent with a language from its class.

Given the assumptions mentioned earlier, none of these constraints are restrictive.

1.2 Time Complexity of Learning Functions

In formal learnability theory there are no a priori constraints on the computational resources required by the learning function. In (Jain et al., 1999) a whole chapter has been devoted to complexity issues in identification, where it is noted that there is a close relationship between the complexity of learning and computational complexity of functionals and operators. Defining the latter is a complex problem and still an active area of research. It is therefore no surprise that only partial attempts have been made at modeling the complexity of the identification process. Some examples are given that are based on bounding the number of mind changes of a learner, or bounding the number of examples required before the onset of convergence. These definitions do not seem to be directly related to any 'computational' notion of complexity. Ideally, such a constraint would satisfy some obvious intuitions about what constitutes tractability: for example, in the worst case a learning function should converge to a correct solution in polynomial time with respect to the size of the input. Such definitions are not directly applicable, since the input is not guaranteed to be helpful, for example it can start with an unbounded number of presentations of the same sentence. In full generality there can never be a bound on the number of time-steps before convergence, so such a constraint poses no bounds on computation time whatsoever.

It turns out that giving a usable definition of the complexity of learning functions is not at all easy. In this subsection some proposals and their problems will be discussed, and the choice for one particular definition will be motivated.

In (Gold, 1967) a definition of efficiency for learning functions known as *text-efficiency* is given: a function φ identifies L *(text-)efficiently* just if there exists no other function that, for every language in L, given the same text, converges at the same point as φ or at an earlier point.

Formally this can be simply regarded as a constraint. Note that this property has nothing to do with the *computational* complexity of learning functions, 'faster' is defined strictly in terms of length of text.

Although the text-efficiency constraint seems to correspond to a rational learning strategy, by itself it is hardly restrictive. Every learnable class is text-efficiently learnable. Also, there is no direct connection between text-efficiency and time complexity. Text-efficiency seems to be of limited interest to the present discussion.[2]

Let the complexity of the update-time of some (computable) learning function φ be defined as the number of computing steps it takes to learn a language, with respect to $|\sigma|$, the size of the input sequence. In (Pitt, 1989) it was first noted that requiring the function to run in

[2]In fact a whole section devoted to this subject in (Osherson et al., 1986) has been completely omitted from the second edition (Jain et al., 1999).

a time polynomial with respect to $|\sigma|$ does not constitute a significant constraint, since one can always define a learning function φ' that combines φ with a clock so that its amount of computing time is bounded by a polynomial over $|\sigma|$. Obviously, φ' learns the same class as φ, and it does so in polynomial update-time.[3]

The problem here is that without additional constraints on φ the 'burden of computation' can be shifted from the number of computations the function needs to perform to the amount of input data considered by the function.[4] Requiring the function to be consistent, that is, requiring that the language associated with its hypothesis always contains $\mathrm{rng}(\sigma)$, already constitutes a significant constraint when used in combination with a complexity restriction (see (Barzdin, 1974)). Some monotone strategies seem to have the same effect. See (Stein, 1998) for a discussion of consistent polynomial-time identification.

In (Angluin, 1979), *consistent and conservative learning with polynomial time of updating conjectures* was proposed as a reasonable criterion for efficient learning. The consistency and conservatism requirements ensure that the update procedure really takes all input into account. It is interesting to note that a conservative (and prudent) learner that is consistent on its class is text-efficient (see Proposition 8.2.2 A, page 172 of (Osherson et al., 1986)). Therefore, the conservative learning functions $\varphi_{k\text{-valued}}$, $\varphi_{\text{least-valued}}$ and $\varphi_{\text{least-card}}$ defined in (Kanazawa, 1998) that are consistent on their class are all text-efficient. This definition was applied in (Arimura et al., 1992) to analyze the complexity of learning a subclass of context-free transformations.

There does not seem to be any generally accepted definition of what constitutes a tractable learning function. A serious problem with Angluin's approach is that it is not generally applicable to learning functions for any given class, since both consistency and (especially) conservatism are restrictive. I will therefore apply only the restrictions of consistency and polynomial update-time, since this seems to be the weakest combination of constraints that is restrictive and has an intuitive relation with standard notions of computational complexity. Even this has drawbacks: not all learnable classes can be learned by a learning function that is consistent on its class, so even this complexity measure cannot be generally applied. There is also no guarantee that for a class that is learnable by a function consistent on that class characteristic samples (i.e. samples that justify convergence to the right grammar) can be given that are uniformly of a size polynomial in the size of their associated grammar.

See (Wiehagen and Zeugmann, 1994), (Wiehagen and Zeugmann, 1995), (Stein, 1998) for discussions of the relation between the consistency constraint and complexity of learning functions.

2 Classical Categorial Grammar and Structure Languages

The classes defined in (Buszkowski, 1987) and (Buszkowski and Penn, 1990) are based on a formalism for (ϵ-free) context-free languages called classical categorial grammar (CCG).[5]. In this section the relevant concepts of CCG will be defined. I will adopt notation from (Kanazawa, 1998)

In CCG each symbol in the alphabet Σ gets assigned a finite number of *types*. Types are constructed from *primitive types* by the operators \ and /. We let Pr denote the set of primitive types. The set of types Tp is defined as follows:

Definition 4 *The set of types* Tp *is the smallest set satisfying the following conditions:*

1. *$Pr \subseteq \mathrm{Tp}$,*
2. *if $A \in \mathrm{Tp}$ and $B \in \mathrm{Tp}$, then $A\backslash B \in \mathrm{Tp}$.*
3. *if $A \in \mathrm{Tp}$ and $B \in \mathrm{Tp}$, then $B/A \in \mathrm{Tp}$.*

One member t of Pr is called the *distinguished type*. In CCG there are only two modes of type combination, *backward application*, $A, A\backslash B \Rightarrow$

[3]To be more precise: in (Daley and Smith, 1986) it was shown that any unbounded monotone increasing update boundary is not by itself restrictive.

[4]Similar issues seem to be important in the field of *computational learning theory* (see (Kearns and Vazirani, 1994) for an introduction). The notion *sample complexity* from this field seems closely related to the notions of text- and data-efficiency. There also exists a parallel with our notion of (polynomial) update-time.

[5]Also known as *AB languages*.

B, and *forward application*, $B/A, A \Rightarrow B$. In both cases, type A is an *argument*, the complex type is a *functor*. *Grammars* consist of type assignments to symbols, i.e. `symbol`$\mapsto T$, where `symbol` $\in \Sigma$, and $T \in \mathrm{Tp}$.

Definition 5 *A* derivation *of B from $A_1, \ldots, A_n$ is a binary branching tree that encodes a proof of $A_1, \ldots, A_n \Rightarrow B$.*

Through the notion of derivation the association between grammar and language is defined. All structures contained in some given structure language correspond to a derivation of type t based solely on the type assignments contained in a given grammar. The *string language* associated with G consists of the strings corresponding to all the structures in its structure language, where the string corresponding to some derivation consists just of the leaves of that derivation.

The class of all categorial grammars is denoted CatG, the grammar system under discussion is $\langle \mathrm{CatG}, \Sigma^{\mathrm{F}}, \mathrm{FL} \rangle$. The symbol FL is an abbreviation of functor-argument language, which is a structure language for CCG. Structures are of the form `symbol`, `fa(s1,s2)` or `ba(s1,s2)`, where `symbol` $\in \mathrm{Pr}$, `fa` stands for forward application, `ba` for backward application and `s1` and `s2` are also structures.

We will only be concerned with structure languages in the remainder of this article. The definition of identification in the limit (Section 1) can be applied in a straightforward way by replacing 'language' with 'structure language', from a formal point of view this makes no difference. Note that, even though structure languages contain more information than string languages, learning a class of structure languages is not necessarily easier than learning the corresponding class of string languages. This is because the *identification criterion* for structure languages is stronger than that for string languages: when learning structure languages, a learner must identify grammars that produce the same *derivations*, not just the same strings. This makes learning such classes hard, from the perspective of both learnability and complexity.

All learning functions in (Kanazawa, 1998) are based on the function GF. This function receives a sample of structures D as input and yields a set of assignments (i.e. a grammar) called the *general form* as output. It is a homomorphism and runs in linear time. It assigns t to each root node, assigns distinct variables to the argument nodes, and computes types for the functor nodes: if symbol `s1`$\mapsto A$, given `ba(s1,s2)` $\Rightarrow B$, `s2`$\mapsto A\backslash B$. If symbol `s1`$\mapsto A$, given `fa(s2,s1)` $\Rightarrow B$, `s2`$\mapsto B/A$.

Categorial types can be treated as terms, so natural definitions of substitution and unification apply. A substitution over a grammar is just a substitution over all of the types contained in its assignments. We state without proof that $\mathrm{FL}(G) \subseteq \mathrm{FL}(\sigma[G])$, see (Kanazawa, 1998) for details.

The following proposition and corollary will be convenient for the proof of NP-hardness:

Proposition 6 *For every structure s, if $s \in \mathrm{FL}(G)$, then there exists a substitution τ such that $\tau[\mathrm{GF}(\{s\})] \subseteq G$.*

`Proof` (Sketch): Since $s \in \mathrm{FL}(G)$, G contains a set of type assignments $G' \subseteq G$ such that G' admits the derivation of type t corresponding with structure s. Each step in a derivation can take the form of just the following three cases:

- The structure deriving type T is symbol $S \in \Sigma$.
- The structure deriving type T is `fa(`$s1, s2$`)`. Structure $s2$ derives some type $Tnew$, structure $s1$ derives $T/Tnew$. The type $Tnew$ may be complex.
- The structure deriving type T is `ba(`$s1, s2$`)`. Structure $s1$ derives some type $Tnew$, structure $s2$ derives $Tnew\backslash T$. The type $Tnew$ may be complex.

This inductive definition shows G' to be equivalent to $\mathrm{GF}(\{s\})$, except that primitive types in $\mathrm{GF}(\{s\})$ may correspond to complex ones in G'. Let τ be the substitution that maps the complex types in G' to the corresponding primitive ones in $\mathrm{GF}(\{s\})$. Then, $\tau[\mathrm{GF}(\{s\})] = G'$. Since $G' \subseteq G$, $\tau[\mathrm{GF}(\{s\})] \subseteq G$ follows. $\square$

Corollary 7 *For every consistent learning function φ learning a subclass of* CatG *and every sequence σ for a language from that subclass there exists a substitution τ such that $\tau[\mathrm{GF}(\sigma)] \subseteq \varphi(\sigma)$.*

Thus, if $\mathrm{GF}(\sigma)$ assigns x different types to the same symbol that are pairwise not unifiable, the consistent learning function $\varphi(\sigma)$ assigns at least x different types to that same symbol.

3 The Classes of Grammars

In the following subsections definitions for the relevant classes will be given. The first two classes are especially important for understanding the proof of NP-hardness.

3.1 Rigid Grammars

A *rigid grammar* is a partial function from Σ to Tp. It assigns either zero or one type to each symbol in the alphabet.

We write $\mathcal{G}_{\text{rigid}}$ to denote the class of rigid grammars over Σ. The class $\{\mathrm{FL}(G) | G \in \mathcal{G}_{\text{rigid}}\}$ is denoted $\mathcal{FL}_{\text{rigid}}$.

This class is learnable with polynomial update-time, by simply unifying all types assigned to the same symbol in the general form. The other classes defined in (Buszkowski, 1987) and (Buszkowski and Penn, 1990) are generalizations of this class.

3.2 k-Valued Grammars

A *k-valued* grammar is a partial function from Σ to Tp. It assigns at most k types to each symbol in the alphabet.

We write $\mathcal{G}_{k\text{-valued}}$ to denote the class of k-valued grammars over Σ. The class $\{\mathrm{FL}(G) | G \in \mathcal{G}_{k\text{-valued}}\}$ is denoted $\mathcal{FL}_{k\text{-valued}}$.

Note that in the special case $k = 1$, $\mathcal{G}_{k\text{-valued}}$ is equivalent to $\mathcal{G}_{\text{rigid}}$.

The learning function φ_{VG_k} learns $\mathcal{G}_{k\text{-valued}}$ from structures. [6]

The proof of NP-hardness that we will give applies directly to the class of k-valued grammars. The proof of this result then applies to some of the following related classes.

3.3 Least-Valued Grammars

A grammar G is called a *least-valued* grammar if it is least-valued with respect to $\mathrm{FL}(G)$.

Let $L \subseteq \Sigma^{\mathrm{F}}$. A grammar $G \in \mathcal{G}_{k+1\text{ -valued}} - \mathcal{G}_{k\text{-valued}}$ is called *least-valued with respect to* L if $L \subseteq \mathrm{FL}(G)$ and there is no $G' \in \mathcal{G}_{k\text{-valued}}$ such that $L \subseteq \mathrm{FL}(G')$.

[6]With this function, and the functions defined for the other classes, we will denote arbitrary learning functions that learn these classes, not necessarily the particular functions defined in (Kanazawa, 1998).

We write $\mathcal{G}_{\text{least-valued}}$ to denote the class of least-valued grammars over Σ. The class $\{\mathrm{FL}(G) \mid G \in \mathcal{G}_{\text{least-valued}}\}$ is denoted $\mathcal{FL}_{\text{least-valued}}$.

The learning function φ_{LVG} learns $\mathcal{G}_{\text{least-valued}}$ from structures.

3.4 Optimal Grammars

Another extension of rigid grammars proposed by Buszkowski and Penn is the class of optimal grammars. The algorithm associated with this class, OG, is based on a generalization of unification called *optimal unification*.

We write $\mathcal{G}_{\text{optimal}}$ to denote the class of optimal grammars over Σ. The class $\{\mathrm{FL}(G) \mid G \in \mathcal{G}_{\text{optimal}}\}$ is denoted $\mathcal{FL}_{\text{optimal}}$.

These grammars can be obtained by unifying $\mathrm{GF}(D)$ 'as much as possible'. Thus, from no $G \in \mathrm{OG}$ a $G' \neq G$ can be obtained by unifying types assigned to the same symbol in G. The class of optimal grammars is not learnable (see (Kanazawa, 1998), Corollary 7.22). It is only mentioned here since it is a superclass of the least cardinality grammars and the minimal grammars.

3.5 Least Cardinality Grammars

We write $\mathcal{G}_{\text{least-card}}$ to denote the class of least cardinality grammars over Σ. The class $\{\mathrm{FL}(G) \mid G \in \mathcal{G}_{\text{least-card}}\}$ is denoted $\mathcal{FL}_{\text{least-card}}$.

If D is a finite set of functor-argument structures, let

$$\mathrm{LCG}(D) = \{G \in \mathrm{OG}(D) | \forall G' \in \mathrm{OG}(D)(|G| \leq |G'|)\}.$$

Let $L \subseteq \Sigma^{\mathrm{F}}$. A grammar G is said to be *of least cardinality with respect to* L if $L \subseteq \mathrm{FL}(G)$ and there is no grammar G' such that $|G'| < |G|$ and $L \subseteq \mathrm{FL}(G')$.

if $G \in \mathrm{LCG}(D)$, then G is of least cardinality with respect to D.

A grammar G is called a *least cardinality grammar* if G is of least cardinality with respect to $\mathrm{FL}(G)$.

The learning function φ_{LCG} learns $\mathcal{G}_{\text{least-card}}$ from structures.

3.6 Minimal Grammars

Like least cardinality grammars, the class of minimal grammars is a subclass of optimal grammars. Hypothesized grammars are required to be minimal according to a certain partial ordering, in addition to being optimal.

We write $\mathcal{G}_{\text{minimal}}$ to denote the class of minimal grammars over Σ. The class $\{\text{FL}(G) \mid G \in \mathcal{G}_{\text{minimal}}\}$ is denoted $\mathcal{FL}_{\text{minimal}}$.

The following proposition will be useful later on:

Proposition 8 *(Kanazawa, 1998) If a grammar G is of least cardinality with respect to L, then G is minimal with respect to L.*

Whether or not $\mathcal{G}_{\text{minimal}}$ is learnable from structures is, as far as we know, still an open question. Kanazawa conjectures it is learnable (see (Kanazawa, 1998), Section 7.3).

4 The Proof

In order to prove NP-hardness of an algorithmic problem L, it suffices to show that there exists a polynomial-time reduction from an NP-complete problem L' to L.[7] We will present such a reduction using the vertex-cover problem, a well-known NP-hard problem from the field of operations research.

Definition 9 *Let $G = (V, E)$ be an undirected graph, where V is a set of vertices and E is a set of edges, represented as tuples of vertices. A* vertex cover *of G is a subset $V' \subseteq V$ such that if $(u, v) \in E$, then $u \in V'$ or $v \in V'$ (or both). That is, each vertex 'covers' its incident edges, and a vertex cover for G is a set of vertices that covers all the edges in E. The* size *of a vertex cover is the number of vertices in it.*

The vertex-cover problem *is the problem of finding a vertex cover of minimum size (called an* optimal vertex cover*) in a given graph.*

The vertex cover problem can be restated as a decision problem*: does a vertex cover of given size k exist for some given graph?*

Proposition 10 *The decision problem related to the vertex-cover problem is* NP*-complete.*

Proposition 11 *The vertex-cover problem is* NP*-hard.*

See (Cormen et al., 1990) for a discussion.

Since the formal proof of Proposition 12 below will be somewhat complex I will first give an informal sketch of its structure. Let graph *Graph* be given. Construct an alphabet A and a sample D, that is, a set of structures $D = \{S_0, \ldots, S_n\}$, using A, following some recipe so that this sample represents *Graph*. A consistent learning function φ presented with D can only conjecture grammars whose associated languages contain D. Using Corollary 7 it will be shown that, in order for these grammars to be in φ's class, they have to correspond to vertex covers for *Graph* of at most some given size. Therefore, computing the conjecture after the last element of D is input solves the decision problem related to the vertex-cover problem, which is NP-complete.[8] Unfortunately, the procedure that converts *Graph* to a sample constructs an alphabet with a size linear in the size of *Graph*. This limits the result to the case where there is no bound on the size of the alphabet.

Proposition 12 *Learning the classes $\mathcal{G}_{k\text{-valued}}$ from structures by means of* one *function that,* for each k, *is responsive and consistent on its class and learns its class prudently, where the alphabet is of unbounded size, is* NP*-hard.*

`Proof:` The decision version of the vertex-cover problem can be transformed in polynomial time to the problem of learning a k-valued grammar from structures by means of a learning function consistent on that class. That is, given a bound on the size of the vertex cover, the function will yield a solution, or will be undefined if no vertex cover of that size exists.[9]

The transformation of the initial graph to an input sample will now be detailed. Edges are numbered $1, \ldots, e$ and vertices are numbered $1, \ldots, v$. First, for every edge i in E, we introduce in the input sample D the structure $\texttt{ba(e,e}_i\texttt{)}$.

Let $\Sigma_1, \Sigma_2, \ldots$ be shorthand for $\texttt{ba(x,v}_1\texttt{)}$, $\texttt{ba(x,ba(x,v}_2\texttt{))}$, ..., respectively. Let the type $X_0^i \backslash \Gamma_i$ be the type assigned to $\texttt{v}_i$ in $\text{GF}(\{\Sigma_i\})$.

[7]This methodology of reductions was introduced in (Karp, 1972), and is also known as many-to-one reduction.

[8]In (Kanazawa, 1998), for each of the classes $\mathcal{G}_{\text{VG}_k}$, $\mathcal{G}_{\text{LVG}}$ and $\mathcal{G}_{\text{LCG}}$ two learning functions are defined, one that is conservative and one that is set-driven. Both are responsive, prudent, and consistent on their class for all these classes, so the proof of Proposition 12 and its corollaries is directly applicable.

[9]Note that this does not mean that the function is not responsive, since it will only be undefined if the input is not from a language from its class.

Note that for any i, j, Γ_i and Γ_j are not unifiable when $i \neq j$.[10]

Add to the sample $\texttt{ba(x,}\Sigma_i\texttt{)}$ for all vertices $1 \leq i \leq v$. For the two vertices $j, k \in V$ incident on edge i, add $\texttt{ba(ba(x,v}_j\texttt{),e}_i\texttt{)}$, $\texttt{ba(ba(x,v}_k\texttt{),e}_i\texttt{)}$.[11]

Let the value of max, which is the size of the desired vertex cover, be assigned to k, the maximum number of types we want to assign to any single symbol in the final conjectured grammar. If $max = 1$, let k be 2. We add to D structures of the same kind as $\Sigma_1, \ldots$ such that some symbols in GF(D) get assigned a number of types that cannot be unified with any other type assigned to the same symbol. This can be done by using a variant on the procedure for creating Σ-types which uses only forward application instead of only backward application. [12] To avoid cluttering the proof these types will be denoted by the (possibly empty) list *Filler*. Add to D structures such that that in GF(D), $max - 2$ (if $max = 1$, let this number be 0) *Filler*-types are assigned to symbols $\texttt{e}_1, \ldots, \texttt{e}_e$, $max - 1$ (if $max = 1$, let this number be 0) *Filler*-types are assigned to symbols $\texttt{v}_1, \ldots, \texttt{v}_v$, and 1 *Filler*-type is assigned to $\texttt{e}$ just if $max = 1$.

To represent graphs in a generic way, some types have indices characteristic for the graph, and some constants characteristic for the graph are also required. Vertex j is connected to k_j edges, which are all edges which are numbered with some e such that $vf_1(e) = ef_j(x)$ or $vf_2(e) = ef_j(x)$, where $1 \leq x \leq k_j$.

Edge e is incident on the two vertices i, j for which $vf_1(e) = ef_i(y)$, for some $1 \leq y \leq k_i$, and $vf_2(e) = ef_j(z)$, for some $1 \leq z \leq k_j$.

Let $G = \text{GF}(D)$:

$$G: \begin{array}{lll}
\texttt{e}_1 & \mapsto & E_1\backslash t, A_{vf_1(1)}\backslash t, A_{vf_2(1)}\backslash t, \mathit{Filler} \\
\ldots & & \\
\texttt{e}_e & \mapsto & E_e\backslash t, A_{vf_1(e)}\backslash t, A_{vf_2(e)}\backslash t, \mathit{Filler} \\
 & & \\
\texttt{e} & \mapsto & E_1, \ldots, E_e, \mathit{Filler} \\
 & & \\
\texttt{v}_1 & \mapsto & X_0^1\backslash\Gamma_1, X_1^1\backslash A_{ef_1(1)}, \ldots, \\
 & & X_{k_1}^1\backslash A_{ef_1(k_1)}, \mathit{Filler} \\
\texttt{v}_2 & \mapsto & X_0^2\backslash\Gamma_2, X_1^2\backslash A_{ef_2(1)}, \ldots, \\
 & & X_{k_2}^2\backslash A_{ef_2(k_2)}, \mathit{Filler} \\
\ldots & & \\
\texttt{v}_v & \mapsto & X_0^v\backslash\Gamma_v, X_1^v\backslash A_{ef_v(1)}, \ldots, \\
 & & X_{k_v}^v\backslash A_{ef_v(k_v)}, \mathit{Filler} \\
 & & \\
\texttt{x} & \mapsto & X_0^1, \ldots, X_{k_1}^1, \\
 & & X_0^2, \ldots, X_{k_2}^2, \\
 & & \ldots, \ldots, \\
 & & X_0^v, \ldots, X_{k_v}^v
\end{array}$$

Suppose this sample D is input for φ_{VG_k}, $k = max$.[13] Then, by Corollary 7, for each $i, 1 \leq i \leq v$, the type $X_0^i\backslash\Gamma_i$ assigned to $\texttt{v}_i$ has to unify with the only types it *can* unify with, which are $X_1^i\backslash A_{ef_i(1)} \ldots X_{k_i}^i\backslash A_{ef_i(k_i)}$. For every such series of unification steps a substitution of the form $\{\Gamma_i \leftarrow A_{ef_i(1)}, \ldots, \Gamma_i \leftarrow A_{ef_i(k_i)}\}$ is obtained.

At this point an index function for the Γ-subtypes in the assignments to $\texttt{e}_1, \ldots, \texttt{e}_e$ is needed, since these unification steps are dependent on the original graph. For this purpose, let the functions $gf_1(i)$ and $gf_2(i)$ denote the two vertices connected to edge i.

These substitutions yield grammar G' (the X-variables are renumbered for readability):

[10]It is easy to see that, using this procedure for generating n such types, this will increase the size of D by a factor only polynomial in n.

[11]We can also allow a single vertex in this set, this would correspond with reflexive connections in the graph. We ignore this possibility for the sake of clarity, since it does not affect the proof in any way.

[12]A proof based on types containing only operator \, or only operator / is desirable since it is more general than a proof based on types containing both operators; such a result would then also hold for unidirectional subclasses of these classes. Using the same procedure for creating the Σ- and *Filler* types creates complications that I have not yet been able to solve.

[13]We show only GF(D) instead of D since D's properties that are relevant to this discussion are much more accessible in this form.

$$G' : \begin{array}{lcl} \mathtt{e}_1 & \mapsto & E_1\backslash t, \Gamma_{gf_1(1)}\backslash t, \Gamma_{gf_2(1)}\backslash t, \mathit{Filler} \\ \ldots & & \\ \mathtt{e}_e & \mapsto & E_e\backslash t, \Gamma_{gf_1(e)}\backslash t, \Gamma_{gf_2(e)}\backslash t, \mathit{Filler} \\ & & \\ \mathtt{e} & \mapsto & E_1, \ldots, E_e, \mathit{Filler} \\ & & \\ \mathtt{v}_1 & \mapsto & X^1\backslash\Gamma_1, \mathit{Filler} \\ \ldots & & \\ \mathtt{v}_v & \mapsto & X^v\backslash\Gamma_v, \mathit{Filler} \\ & & \\ \mathtt{x} & \mapsto & X^1, \ldots, X^v \end{array}$$

Now, in order to obtain a grammar that is k-valued ($k = max$), we need to unify two of the types assigned to $\mathtt{e}_i$, for all i. Since the Γ-types are not unifiable, this means that either $E_i\backslash t$ and $\Gamma_{gf_1(i)}\backslash t$, or $E_i\backslash t$ and $\Gamma_{gf_2(i)}\backslash t$ have to be unified. This will result either in the substitution $\{\Gamma_{gf_1(i)} \leftarrow E_i\}$ or in the substitution $\{\Gamma_{gf_2(i)} \leftarrow E_i\}$. Since $\mathtt{e} \mapsto E_1, \ldots, E_e$, this results in the assignment of either $\Gamma_{gf_1(i)}$ or $\Gamma_{gf_2(i)}$ to $\mathtt{e}$.

This unification step is intended to correspond to including vertex $gf_1(i)$ or $gf_2(i)$ in the vertex-cover.

At this point another index function is needed, this time for the Γ-types assigned to $\mathtt{e}$. For this purpose, let the functions $gef(1), \ldots, gef(max)$ denote the vertices in the vertex cover.

The final output of φ_{VG_k}, if it is defined, is G'':

$$G'' : \begin{array}{lcl} \mathtt{e}_1 & \mapsto & \Gamma_{gf_1(1)}\backslash t, \Gamma_{gf_2(1)}\backslash t, \mathit{Filler} \\ \ldots & & \\ \mathtt{e}_e & \mapsto & \Gamma_{gf_1(e)}\backslash t, \Gamma_{gf_2(e)}\backslash t, \mathit{Filler} \\ & & \\ \mathtt{e} & \mapsto & \Gamma_{gef(1)}, \ldots, \Gamma_{gef(max)}, \mathit{Filler} \\ & & \\ \mathtt{v}_1 & \mapsto & X^1\backslash\Gamma_1, \mathit{Filler} \\ \ldots & & \\ \mathtt{v}_v & \mapsto & X^v\backslash\Gamma_v, \mathit{Filler} \\ & & \\ \mathtt{x} & \mapsto & X^1, \ldots, X^v \end{array}$$

Whether or not all types assigned to $\mathtt{x}$ are unified has no consequence for the structure language.

The resulting grammar can be read as a solution by taking the set S of all the Γ-types assigned to $\mathtt{e}$, and adding vertex v to the solution for each $\mathtt{v}_i$ that has type $X^i\backslash\Gamma_i$, $\Gamma_i \in S$, assigned to it.

Since both the conversion from graph to input sample and the conversion from resulting grammar to set of vertices can be done in polynomial time, the learning function has to be NP-hard. This implies that its update-time is NP-hard, since its total computation time is

$$\sum_{n=1}^{\mathrm{size}(D)} \text{update-time for } n^{\mathrm{th}} \text{ element in } D,$$

size(D) is polynomial in the size of the graph, as is the size of each element in D.

Any grammar output by such a function that is k-valued, $k = max$, will look like G''. Since such a grammar will correspond to a vertex cover any function that can learn any of these classes prudently and is responsive and consistent on that class will be able to solve the decision problem related to the vertex-cover problem after a polynomial-time reduction. □

Corollary 13 *(Of the proof) Learning $\mathcal{G}_{\text{least-valued}}$ from structures by means of a function that is responsive and consistent on its class and learns its class prudently, where the alphabet is of unbounded size, is* NP*-hard.*

Obviously, exactly the same proof works for learning $\mathcal{G}_{\text{least-valued}}$, since, because of the introduction of the *Filler*-types, there cannot be any grammars obtained from D with $k < max$, so the least value for k is max.

Corollary 14 *(Of the proof) Learning $\mathcal{G}_{\text{least-card}}$ from structures by means of a function that is responsive and consistent on its class and learns its class prudently, where the alphabet is of unbounded size, is* NP*-hard.*

The proof works for learning $\mathcal{G}_{\text{least-card}}$, since the k-valued grammar obtained by learning $\mathcal{G}_{k\text{-valued}}$ is optimal (all symbols have k non-unifiable types assigned, recall the remark concerning symbol $\mathtt{x}$), and all optimal grammars obtainable from D have the same cardinality.

The proof of Proposition 12 cannot be used for $\mathcal{G}_{\text{minimal}}$. However, the relation between $\mathcal{G}_{\text{minimal}}$ and $\mathcal{G}_{\text{least-card}}$ provides a different route for proving NP-hardness.

Let φ be a computable function for a class $\mathcal{L}$ that learns $\mathcal{L}$ consistently. Then the learning function φ' for a class $\mathcal{L}', \mathcal{L} \subseteq \mathcal{L}'$ that learns $\mathcal{L}'$ consistently has a time complexity that is the same as, or worse than, the time complexity of φ. From this and Proposition 8 the following proposition is straightforward:

Proposition 15 *Learning $\mathcal{G}_{\text{minimal}}$ by means of a function that is responsive and consistent on $\mathcal{G}_{\text{minimal}}$ and learns $\mathcal{G}_{\text{minimal}}$ prudently, where the size of the alphabet is unbounded, is* NP-*hard.*

A proof of NP-hardness gives evidence for the intractability of a problem. After such a proof has been given it is natural to ask whether such a problem is NP-*complete*. In order to prove NP-*completeness* of a problem L that has been shown to be NP-hard, one needs to show that $L \in \text{NP}$. This would imply that there exists an algorithm that verifies solutions for L in polynomial time. Normally this is the 'easy' part of an NP-completeness proof.

In this case, however, it is not at all clear what such algorithms are supposed to do, let alone whether they exist. Their task, among other things, is checking whether the grammar is consistent with the input sequence, whether it is in the right class, and whether the grammar is justified in giving its conjecture. Obviously the last task is the most problematic.

Checking consistency is polynomial in $|D|$ (since membership is decidable in polynomial time for context-free structure languages), but it is not even clear whether for all φ learning any of the classes under discussion, $|D|$ may be exponential in $|G|$ for some G in φ's class.

Checking whether a grammar is k-valued, or optimal, can obviously be done in polynomial time, but even checking whether grammar G can be derived from grammar G' by unification may not be so simple. Defining this criterion and proving existence of a polynomial time verification algorithm is expected to be much harder than the proof of Proposition 12.

An interesting question is whether there exist (non-trivial) learnable subclasses of the classes under discussion for which polynomial-time consistent learning algorithms *do* exist.[14] A necessary (but not sufficient) condition for such a class would be that vertex-cover problems cannot be recast as learning problems in polynomial time. It is easy to see that this requires a class definition that is not (crucially) based on the number of type assignments in the grammar.

[14] Obviously, consistently learning any *superclass* of the classes under discussion is an NP-hard problem.

5 Conclusion and Further Research

In this paper it is shown that learning any of the classes $\mathcal{G}_{\text{least-valued}}$, $\mathcal{G}_{\text{least-card}}$, and $\mathcal{G}_{\text{minimal}}$ from structures by means of a learning function that is consistent on its class is NP-hard. The result for the classes $\mathcal{G}_{k\text{-valued}}$ is weaker: one function that can learn these classes *for each k* and is consistent on its class is NP-hard. It is an open question whether there exist polynomial-time learning functions for $\mathcal{G}_{k\text{-valued}}$ for each k separately, although I feel it is unlikely. Showing intractability for $k = 2$ would imply intractability for all $k > 1$, since $\mathcal{G}_{k\text{-valued}} \subseteq \mathcal{G}_{k+1\text{-valued}}$. Note that these results hold just under the assumption that there is no bound on the size of the alphabet. It is an open question whether there exists a proof with an alphabet of some constant size.

It is a well-known fact that learning functions for any learnable class without consistency- and monotonicity constraints can be transformed to learning functions that have polynomial update-time using a trivial procedure (see Subsection 1.2). It is an open question whether there exist 'intelligent' inconsistent learning functions that have polynomial update-time for the classes under discussion.

Since the relation between structure language and string language is so clear-cut, it is in general easy to transfer results from one to the other. In (Kanazawa, 1998) some results concerning learnability of classes of structure languages were used to obtain learnability results for the corresponding classes of string languages. It might be possible to do the same with complexity results, i.e. obtain an NP-hardness result for learning $\mathcal{G}_{\text{least-valued}}$ from strings, for example.

Note that the proof of Proposition 15 nicely demonstrates that complexity results can be obtained even for classes for which learnability is still an open question.

The proof of Proposition 12 relies on a subclass of languages that can all be identified with sequences that have a length polynomial in the size of their associated grammars. This is not necessarily true for any arbitrary language in the class, so data-complexity issues may make the complexity of learning these classes even worse than Proposition 12 suggests.

Instead of investigating the complexity of learning for each distinct class on an individual basis, it would be nice to have insights into the direct relation between complexity and some structural properties of learnable classes. This would be an interesting topic for future research.

Analyzing these classes in terms of intrinsic complexity (see (Freivalds et al., 1995)) would yield insights into the relation between these and other classes, and into the structure of the complexity hierarchy of learnable classes in general.

References

D. Angluin. 1979. Finding common patterns to a set of strings. In *Proceedings of the 11th Annual Symposium on Theory of Computing*, pages 130–141.

D. Angluin. 1980. Finding patterns common to a set of strings. *Journal of Computer System Sciences*, 21:46–62.

Hiroki Arimura, Hiroki Ishizaka, and Takeshi Shinohara. 1992. Polynomial time inference of a subclass of context-free transformations. In *Proceedings of the Fifth Annual ACM Workshop on Computational Learning Theory*, pages 136–143, Pittsburgh, Pennsylvania, 27–29 July. ACM Press.

J. Barzdin. 1974. Inductive inference of automata, functions and programs. In *Proceedings International Congres of Math.*, pages 455–460, Vancouver.

W. Buszkowski and G. Penn. 1990. Categorial grammars determined from linguistic data by unification. *Studia Logica*, 49:431–454.

W. Buszkowski. 1987. Discovery procedures for categorial grammars. In E. Klein and J. van Benthem, editors, *Categories, Polymorphism and Unification.* University of Amsterdam.

Thomas H. Cormen, Charles E. Leiserson, and Ronald L. Rivest. 1990. *Introduction to Algorithms.* MIT Press, Cambridge, Mass., eighteenth edition.

R. Daley and C. Smith. 1986. On the complexity of inductive inference. *Information and Control*, 69:12–40.

R. Freivalds, E. Kinber, and C. Smith. 1995. On the intrinsic complexity of learning. In Paul Vitányi, editor, *Second European Conference on Computational Learning Theory*, volume 904 of *Lecture Notes in Artificial Intelligence*, pages 154–168. Springer-Verlag.

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

Sanjay Jain, Daniel Osherson, James Royer, and Arun Sharma. 1999. *Systems that Learn: An Introduction to Learning Theory.* The MIT Press, Cambridge, MA., second edition.

M. Kanazawa. 1998. *Learnable Classes of Categorial Grammars.* CSLI Publications, Stanford University.

Richard M. Karp. 1972. Reducibility among combinatorial problems. In Raymond E. Miller and James W. Thatcher, editors, *Complexity of Computer Computations.* Plenum Press.

Michael J. Kearns and Umesh V. Vazirani. 1994. *An Introduction to Computational Learning Theory.* Cambridge, Mass.: MIT Press.

D. N. Osherson, M. Stob, and S. Weinstein. 1986. *Systems that Learn: An Introduction to Learning Theory for Cognitive and Computer Scientists.* MIT Press, Cambridge, MA.

D. N. Osherson, D. de Jongh, E. Martin, and S. Weinstein. 1997. Formal learning theory. In *(van Benthem and ter Meulen, 1997).* Elsevier Science B.V.

L. Pitt. 1989. Inductive inference, dfas, and computational complexity. In K. P. Jantke, editor, *Proceedings of International Workshop on Analogical and Inductive Inference*, number 397 in Lecture Notes in Computer Science, pages 18–44.

Werner Stein. 1998. Consistent polynominal identification in the limit. In *Algorithmic Learning Theory (ALT)*, volume 1501 of *Lecture Notes in Computer Science*, pages 424–438, Berlin. Springer-Verlag.

J. van Benthem and A. ter Meulen, editors. 1997. *Handbook of Logic and Language.* Elsevier Science B.V.

R. Wiehagen and T. Zeugmann. 1994. Ignoring data may be the only way to learn efficiently. *Journal of Experimental and Theoretical Artificial Intelligence*, 6:131–144.

R. Wiehagen and T. Zeugmann. 1995. Learning and consistency. In K. P. Jantke and S. Lange, editors, *Algorithmic Learning for Knowledge-Based Systems*, Lecture Notes in Artificial Intelligence 961, pages 1–24. Springer-Verlag.

Keith Wright. 1989. Identification of unions of languages drawn from an identifiable class. In *The 1989 Workshop on Computational Learning Theory*, pages 328–333. San Mateo, Calif.: Morgan Kaufmann.