JMLR: Workshop and Conference Proceedings 34:49–63, 2014 Proceedings of the 12th ICGI

A bottom-up efficient algorithm learning substitutable languages from positive examples

François Coste FRANCOIS.COSTE@INRIA.FR
Gaëlle Garet GAELLE.GARET@INRIA.FR
Jacques Nicolas JACQUES.NICOLAS@INRIA.FR
Irisa / Inria Rennes - Bretagne Atlantique
Campus universitaire de Beaulieu
35042 Rennes, France

Editors: Alexander Clark, Makoto Kanazawa and Ryo Yoshinaka

Abstract

Based on Harris's substitutability criterion, the recent definitions of classes of substitutable languages have led to interesting polynomial learnability results for expressive formal languages. These classes are also promising for practical applications: in natural language analysis, because definitions have strong linguisitic support, but also in biology for modeling protein families, as suggested in our previous study introducing the class of local substitutable languages. But turning recent theoretical advances into practice badly needs truly practical algorithms. We present here an efficient learning algorithm, motivated by intelligibility and parsing efficiency of the result, which directly reduces the positive sample into a small non-redundant canonical grammar of the target substitutable language. Thanks to this new algorithm, we have been able to extend our experimentation to a complete protein dataset confirming that it is possible to learn grammars on proteins with high specificity and good sensitivity by a generalization based on local substitutability.

Keywords: (local) substitutable languages, learning algorithm, canonical grammar, proteins

1. Introduction

Based on Harris's substitutability criterion, the recent definitions of classes of context-free substitutable languages have led to interesting polynomial learnability results for expressive formal languages. These classes are also promising for practical applications: in natural language from the linguistics motivation of the definitions but also in biology to learn grammars modeling protein families, as suggested in our previous study introducing the class of local substitutable languages. To turn theory into practice, we present here an efficient learning algorithm, motivated by intelligibility and parsing efficiency of the result, which directly reduces the positive sample into a small non-redundant canonical grammar of the target substitutable language. We introduce first the definitions of substitutable languages and a first algorithm learning them in section 2, before introducing our algorithm in section 3 and showing its interest experimentally on artificial and applicative datasets in section 4.

2. Learning substitutable languages

2.1. Formal languages and substitutability

We introduce briefly here the definitions and notations related to substitutable languages.

Let Σ be a non-empty finite set of atomic symbols. Σ^* is the set of all finite strings over the alphabet Σ. We denote the length of a string x by $|x|$, the empty string by λ, the set of non-empty strings $\Sigma^* \setminus \{\lambda\}$ by Σ^+ and the set of strings of length k $\{x\colon |x| = k\}$ by Σ^k. For a language $L \subseteq \Sigma^*$, the set of its substrings is $Sub(L) = \{y \in \Sigma^*\colon x, z \in \Sigma^*, xyz \in L\}$ and the set of its contexts is $Con(L) = \{\langle x, z\rangle \in \Sigma^* \times \Sigma^*\colon y \in \Sigma^*, xyz \in L\}$. The empty context is $\langle\lambda, \lambda\rangle$.

The distribution of a string $y \in \Sigma^*$ with respect to a language L is defined to be its set of contexts in L: $D_L(y) = \{\langle x, z\rangle \in \Sigma^* \times \Sigma^*\colon xyz \in L\}$. Two strings y_1 and y_2 in Σ^* are syntactically congruent for a language L, denoted $y_1 \equiv_L y_2$, iff $D_L(y_1) = D_L(y_2)$. The equivalence relation $\equiv_L$ defines a congruence on the monoid Σ^* since straightforwardly $y_1 \equiv_L y_2$ implies $\forall x, z$ in $\Sigma^*, xy_1z \equiv_L xy_2z$. We denote the congruence class of y by $[y]_L = \{y' \in \Sigma^*\colon y \equiv_L y'\}$ and the concatenation of two languages L_1 and L_2 by $L_1L_2 = \{y_1y_2\colon y_1 \in L_1, y_2 \in L_2\}$. The congruence class $[\lambda]$ is called the *unit congruence class*. The set $\{y\colon D_L(y) = \emptyset\} = \Sigma^* \setminus Sub(L)$, when non-empty, is called the *zero congruence class*. A congruence class is non-zero if it is a subset of $Sub(L)$. Note that for any strings y_1, y_2 in Σ^*, $[y_1y_2]_L \supseteq [y_1]_L[y_2]_L$.

Interested by learnability of natural languages and inspired by distributional learning, Clark and Eyraud (2007) introduced the substitutable languages as a simple formal class of languages based on Harris's substitutability criterion. Two non-empty strings y_1 and y_2 are said to be *weakly substitutable* in a language L, denoted $y_1 \doteq_L y_2$, iff there exists $x, z \in \Sigma^*$, such that $xy_1z \in L \wedge xy_2z \in L$. Substitutable languages are those such that weak substitutability implies syntactic congruence, *i.e.* such that $y_1 \doteq_L y_2$ implies $y_1 \equiv_L y_2$ (or equivalently, such that $D_L(y_1) \cap D_L(y_2) \neq \emptyset$ implies $D_L(y_1) = D_L(y_2)$). At the word level, we get the following definition:

Definition 1 *(Clark and Eyraud, 2007) A language L is* substitutable *iff for any $x_1, y_1, z_1, x_2, y_2, z_2 \in \Sigma^*, y_1, y_2 \neq \lambda$:*

$$x_1y_1z_1 \in L \wedge x_1y_2z_1 \in L \Rightarrow (x_2y_1z_2 \in L \Leftrightarrow x_2y_2z_2 \in L).$$

Carrying on an analogy between substitutability for context-free languages and reversibility introduced by Angluin (1982) for regular languages, Yoshinaka (2008) introduced the hierarchy of k, l-substitutable context-free language as the counterpart of the k-reversible hierarchy for regular languages. In this hierarchy, expressiveness increases with the parameters k and l by restricting substitutability to occur only in same right and left contexts of respective length k and l. Just as for substitutable languages, one can define two non-empty strings y_1 and y_2 to be *weakly k, l-substitutable in context* $\langle u, v\rangle \in \Sigma^k \times \Sigma^l$ in a language L, denoted $uy_1v \doteq_L uy_2v$, iff there exists $x, z \in \Sigma^*$, such that $xuy_1vz \in L \wedge xuy_2vz \in L$. The k, l-substitutable languages are those such that weak substitutability in a context from $\Sigma^k \times \Sigma^l$ implies syntactic congruence in the same context, *i.e.* such that $uy_1v \doteq_L uy_2v$ implies $uy_1v \equiv_L uy_2v$ (or equivalently, such that $D_L(uy_1v) \cap D_L(uy_2v) \neq \emptyset$ implies $D_L(uy_1v) = D_L(uy_2v)$):

Definition 2 *(Yoshinaka, 2008) A language L is* k,l-substitutable *iff for any* $x_1, y_1, z_1, x_2, y_2, z_2 \in \Sigma^*, u \in \Sigma^k, v \in \Sigma^l, uy_1v, uy_2v \neq \lambda$:

$$x_1uy_1vz_1 \in L \wedge x_1uy_2vz_1 \in L \Rightarrow (x_2uy_1vz_2 \in L \Leftrightarrow x_2uy_2vz_2 \in L).$$

The definition of the substitutable strings y_1 and y_2 above are based on common global contexts. In Coste et al. (2012), we introduced for the characterization of proteins a less stringent criterion based on common *local* contexts that applies to more practical cases. Introducing the parameters k and l to set the minimal length of the left and right local contexts, two non-empty strings y_1 and y_2 are said *weakly* (k,l)*-local substitutable* in a language L, denoted $y_1 \doteq_L^{k,l} y_2$, iff there exists $x_1, x_2, z_1, z_2 \in \Sigma^*, u \in \Sigma^k, v \in \Sigma^l$, such that: $x_1uy_1vz_1 \in L \wedge x_2uy_2vz_2 \in L$. The k,l-local substitutable languages are those such that weak (k,l)-local substitutability implies syntactic congruence, *i.e.* such that $y_1 \doteq_L^{k,l} y_2$ implies $y_1 \equiv_L y_2$ (or equivalently if we define $D_L^{k,l}(y)$ as $\{\langle u, v\rangle \in \Sigma^k \times \Sigma^l : uyv \in Sub(L)\}$, such that $D_L^{k,l}(y_1) \cap D_L^{k,l}(y_2) \neq \emptyset$ implies $D_L(y_1) = D_L(y_2)$):

Definition 3 *(Coste et al., 2012) A language L is* k,l-local substitutable *iff for any* $x_1, y_1, z_1, x_2, y_2, z_2, x_3, z_3 \in \Sigma^*, u \in \Sigma^k, v \in \Sigma^l, uy_1v, uy_2v \neq \lambda$:

$$x_1uy_1vz_1 \in L \wedge x_3uy_2vz_3 \in L \Rightarrow (x_2y_1z_2 \in L \Leftrightarrow x_2y_2z_2 \in L).$$

And by crossing the local extension with the extension by Yoshinaka (2008), we get the definition of the k,l-local contextually substitutable languages:

Definition 4 *(Coste et al., 2012) A language L is* k,l-local contextually substitutable *iff for any* $x_1, y_1, z_1, x_2, y_2, z_2, x_3, z_3 \in \Sigma^*, u \in \Sigma^k, v \in \Sigma^l, uy_1v, uy_2v \neq \lambda$:

$$x_1uy_1vz_1 \in L \wedge x_3uy_2vz_3 \in L \Rightarrow (x_2uy_1vz_2 \in L \Leftrightarrow x_2uy_2vz_2 \in L).$$

The class of substitutable languages defined by Clark and Eyraud (2007) stands at the limit of the hierarchy of k,l-local substitutable languages when k and l tend to infinity, and conversely is the first class in the hierarchy of k,l-substitutable languages, with k and l equal to 0. As for reversible languages, we will thus use the term *zero-substitutable language* when we want to designate specifically this class and we will keep *substitutable languages* as a generic term for the classes of languages formally defined with respect to a substitutability criterion.

In the next sections, we will consider the problem of learning substitutable languages with algorithms using *context-free grammar* representations. A context-free grammar is defined classically as a quadruple $G = \langle \Sigma, N, S, P\rangle$ where Σ is a finite alphabet of terminal symbols, N is a finite alphabet of variables or non-terminals, $S \in N$ is the start symbol and P is a finite set of production rules of the form $N \times (N \cup \Sigma)^+$. We say that a sequence $\delta A\gamma$ from $(N \cup \Sigma)^+$ can be derived into $\delta\alpha\gamma$, denoted $\delta A\gamma \Rightarrow_G \delta\alpha\gamma$ if there exists a production rule $A \to \alpha$ in P. The transitive closure of $\Rightarrow_G$ is denoted $\overset{+}{\Rightarrow}_G$ and its reflexive transitive closure $\overset{*}{\Rightarrow}_G$. The set of strings that can be derived from S is the set of sentential forms $\{\alpha \in (N \cup \Sigma)^* : S \overset{*}{\Rightarrow}_G \alpha\}$ and the context-free language defined by a context-free grammar G is the set of sentential forms consisting entirely of terminal symbols $L(G) = \{w \in \Sigma^* : S \overset{*}{\Rightarrow}_G w\}$. Let us note that from our definition of production rules, the empty string cannot be derived from G. In that case, a context-free grammar G is in Chomsky Normal Form (CNF) iff its productions are of the form $A \to BC$, $A \to a$, where $A, B, C \in N$ and $a \in \Sigma$.

2.2. A first grammatical inference algorithm for substitutable languages

For our experiments on grammatical inference of k, l-local substitutable languages in Coste et al. (2012), we implemented the algorithm 1 that is a straightforward adaptation of the SGL algorithm introduced for substitutable languages by Clark and Eyraud (2007).

Algorithm 1: SGL_{LS} (Substitution Graph Learner for k, l-local substitutable languages)

Input: Set of sequences K on alphabet Σ, int k, int l
Output: Grammar $G = \langle \Sigma, N_K, S_K, P_K \rangle$
`/* Partition` $Sub(K)$ `in substitutability classes */`
$\mathcal{C}_K \leftarrow$ `Local_substitutability_classes`(K, k, l)
$\forall y \in Sub(K), C_K(y) = C \in \mathcal{C}_K : y \in C$
`/* Build grammar */`
$N_K \leftarrow \emptyset, P_K \leftarrow \emptyset$
for $C \in \mathcal{C}_K$ **do**
  `/* A non-terminal for each substitutability class */`
  $N_K \leftarrow N_K \cup \{[\![C]\!]\}$
  if $C \cap K \neq \emptyset$ **then**
    $S_K \leftarrow [\![C]\!]$
  `/* Productions rules for each substring in the class */`
  for $y \in C$ **do**
    if $|y| > 1$ **then**
      `/* Branching rules: a 'CNF' rule for each split */`
      for $y_1 \in \Sigma^+, y_2 \in \Sigma^+ : y_1 y_2 = y$ **do**
        $P_K \leftarrow P_K \cup \{[\![C]\!] \rightarrow [\![C_K(y_1)]\!][\![C_K(y_2)]\!]\}$
    else
      `/* Terminal rule */`
      $P_K \leftarrow P_K \cup \{[\![C]\!] \rightarrow y\}$
return $\langle \Sigma, N_K, S_K, P_K \rangle$

Algorithm 2: *Local_substitutability_classes*

Input: Set of sequences K on alphabet Σ, int k, int l
Output: k, l local substitutability classes on K
`/* Build substitutability graph on substrings */`
$V \leftarrow \{y \in \Sigma^+ : y \in Sub(K)\}$
$E \leftarrow \{\{y_1, y_2\} \in V \times V : uy_1v \in Sub(K), uy_2v \in Sub(K), y_1 \neq y_2, u \in \Sigma^k, v \in \Sigma^l\}$
`/* Return connected components of graph */`
return `Connected_components`$(\langle V, E \rangle)$

Given a set of sequences K and parameters k and l specifying the target class of local substitutable languages, this algorithm partitions all the substrings of K into k, l-local substitutability classes in K using a substitution graph and returns a CNF context-free grammar. This grammar contains a *non-terminal symbol* $[\![C]\!]$ for each substitutability class C, *branching rules*, carrying on the generalization, $[\![C_K(y_1y_2)]\!] \rightarrow [\![C_K(y_1)]\!][\![C_K(y_2)]\!]$ for

each possible concatenation y_1y_2 that results in a substring of K — where $C_K(x)$ identifies the substitutability class of x — and *terminal rules* that generate terminal symbols.

Let us remark that this algorithm may be adapted to other classes of substitutable languages, simply by changing the function returning the substitutability classes. This can be achieved by modifying in line 2 of algorithm 2 the substrings in relation in the substitution graph. The definition of the set of edges was replaced for substitutable languages by: $E \leftarrow \{\{y_1, y_2\} \in V \times V : xy_1z \in K, xy_2z \in K, y_1 \neq y_2, x \in \Sigma^*, z \in \Sigma^*\}$, for k, l-substitutable languages by: $E \leftarrow \{\{uy_1v, uy_2v\} \in V \times V : xuy_1vz \in K, xuy_2vz \in K, y_1 \neq y_2, x \in \Sigma^*, z \in \Sigma^*, u \in \Sigma^k, v \in \Sigma^l\}$ and for k, l-local context substitutable languages by: $E \leftarrow \{\{uy_1v, uy_2v\} \in V \times V : uy_1v \in Sub(K), uy_2v \in Sub(K), y_1 \neq y_2, u \in \Sigma^k, v \in \Sigma^l\}$. This can be done in all the algorithms presented in this paper and we can thus restrict ourselves hereafter to the k, l-local substitutability case to simplify the presentation without loss of generality.

Even if it does not always return a grammar from the target class of languages, this kind of algorithm enables to identify a target grammar G in polynomial time from a characteristic sample of polynomial size in $|G|.\tau_G$, where τ_G is the thickness of the grammar, provided that its context-free substitutability class is known (Clark and Eyraud, 2007; Yoshinaka, 2008). But even if the algorithm is polynomial, it cannot be used in practice on real data sets as witnessed by our first experiments on proteins (Coste et al., 2012), since it requires hours for each run of the algorithm preventing significant cross-validation and more systematic studies on the parameters' influence. As a matter of fact, the resulting grammars suffer from embedding too many ambiguities and redundancies. From a parsing perspective or even with the simple goal of manually checking and interpreting the rules, the obtained grammars are too large, an issue that affects also clearly the learning run time.

Hence, motivated by speeding up the experiments and driven by legibility and parsing efficiency of the grammar, we have designed a new algorithm, implemented in Fall 2012 but still unpublished, learning directly grammars in a minimal non-redundant form. The next section describes this new algorithm.

3. Learning reduced grammars

3.1. Reduced Grammars

To improve parsing time and improve the legibility of grammars learnt by SGL_{LS} as well as their derivation trees, the Chomsky Normal Form can be transformed into a *reduced form* that avoids unnecessary derivation steps and redundancies by:

- removing unnecessary non-terminals whose derivation is deterministic(if a non-terminal A is the left-hand side of only one rule $A \rightarrow \alpha$, the non-terminal and the rule are deleted and each occurrence of A in the other rules is replaced by α);
- minimizing the right-hand sides of the production rules (replacing any substring β of the right-hand side by the smallest substrings α in $N \cup \Sigma$ from which β can be derived)

The first reduction corresponds to removing non-terminals that give rise to *vacuous local derivation trees* (Clark, 2011). We called this kind of congruence classes *composite* since they can be factorized into the concatenation of a set of congruence classes. Formally:

Definition 5 (Composite congruence class) *Let a language L whose set of non-zero and non-unit congruence classes is C^+. A class $[y] \in C^+$ is* composite *for L iff:*

$$\exists [x_1], \dots [x_m] \in C^+, m \geq 2, [y] = [x_1] \dots [x_m]$$

A congruence class is *prime* if it is not composite. These classes are the ones to be kept as non-terminals. In the algorithms, we use a more operational characterization of prime (and composite) congruence classes which can be deduced from syntactic congruence properties:

Lemma 6 *Let a language L whose set of non-zero and non-unit congruence classes is C^+. A class $[y]$ in C^+ is prime for L iff $\forall y_1 y_2 \in [y], [y] \not\subset [y_1][y_2]$.*

Proof This is a simple application of the fact that the contexts of a string include the concatenation of its substring contexts. Any equation involving $m > 2$ substrings is thus superseded by the equation involving only the first substring and the concatenation of the other substrings. Moreover, the equation can be replaced by an inclusion since the other direction is already granted. ■

The second reduction, since non-terminals and the languages they generate are fixed, can be handled as an extension of minimal grammar parsing (Carrascosa et al., 2011): each non-terminal generates a whole set of substitutable words rather than a single word. It can be solved similarly by dynamic programming on graphs representing right-hand side's strings of the rules, with the added difficulty that all non-redundant paths have to be found. Solving this problem is equivalent to searching for each right-hand side α the set of non-redundant strings generating all the sequences generated from α (see next section for details). Note that this algorithm results in the same set of rules than the algorithm recently proposed by Clark (2014) even if approaches differ: building directly the set of non-redundant rules by dynamic programming in our case versus filtering out from the whole set of potential valid rules the so-called *pleonastic* rules. Based on the 'fundamental lemma' of substitutable languages proven by Clark (2014), the *reduced form* computed by this algorithm is thus also a *canonical form* of the grammar for substitutable languages.

So far we have only discussed the reduction of the grammar into a minimal non-redundant canonical form. We present in the next section the algorithm ReGLiS based on the same dynamic programming ideas but learning simultaneously this reduced form and the content of the substitutability classes by a bottom-up reduction of the sample.

3.2. Learning reduced grammars: ReGLiS

The ReGLiS algorithm is detailed in algorithm 3. Taking as input a sample of strings K on an alphabet Σ, and two parameters k and l defining the target class of local substitutable languages, ReGLiS returns a grammar in reduced form of the target language when K includes a characteristic sample of the language for SGL_{LS}.

Like SGL_{LS} presented in section 2.2, ReGLiS builds a grammar based on a set of equivalence classes $\mathcal{C}_K$ defined from weak substitutability evidence in K. Let us remark that the rest of the algorithm is independent from the actual targeted class of languages:

the classes are based here on k, l-local substitutability, but other weak substitutability criteria can be used to adapt the algorithm to other classes of substitutable languages.

In contrast with SGL_{LS} introducing a non-terminal for each equivalence class, ReGLiS takes care to introduce a non-terminal only for the substitutability classes containing more than one string and satisfying Lemma 6 on $\mathcal{C}_K$ which we call *K-prime* substitutability classes. The symbol introduced for the K-prime substitutability class containing K is the start symbol. For each of these non-terminals, the set of production rules contains as many rules as strings in its substitutability class, deriving each in one step the non-terminal into one of these strings. The grammar built this way is only able to generate the strings from K and contains also unreachable rules linking each non-terminal with the substrings of its substitutability class.

A generalization step at the core of the algorithm is then achieved by a bottom-up rewriting process on the right-hand sides, starting by the shortest ones and optimizing the set of branching rules by dynamic programming on a parsing graph for each right-hand side of rule.

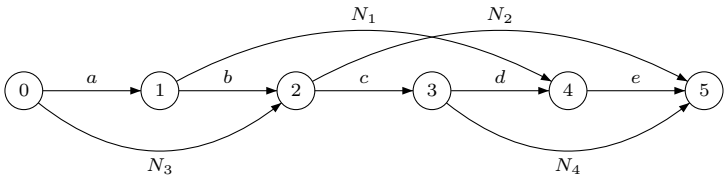


Figure 1: Example of parsing graph for *abcde*: non-redundant right-hand sides are aN_1e and N_3N_2.

Specifically, for each right-hand side α, a *parsing graph* is built by algorithm 4 (see example in figure 1). Intuitively, the graph represents all the possible generation of substrings from α by substitutability class non-terminals. Finding all non-redundant most general right-hand sides allowing to generate α, is done by algorithm 5. Forgetting the symbols, this algorithm focuses on paths in the parsing graph. Paths are represented as sequences of vertices, and adding a vertex i at the end of a path π is denoted by the concatenation $\pi.i$. A path $\pi_2 = t_1 \dots t_m$ is reducible in $\pi = s_1 \dots s_n$, denoted $\pi \prec_r \pi_2$, if π is a strict substring of π_2 that has the same beginning and end than π_2, *i.e.* $s_1 = t_1$ and $s_n = t_m$. A path π is said irreducible if it is minimal for the partial order $\prec_r$. Algorithm 5 implements a dynamic programming search of all irreducible paths in the parsing graph and return then all the non-redundant right-hand side generalizing α to be used to replace it.

Two improvements have been included in the algorithm to enable also the identification of the target language for some samples on which SGL_{LS} would not succeed. First, if during the reduction one gets an already existing right-hand side in the set of production rules, the corresponding non-terminals are unified to avoid generating the same substring by two different non-terminals and still ensure the transitivity of the substitution classes. Second, since languages recognized by non-terminals are increasing, new edges can appear in the

parsing graph despite the ordering of the right-hand sides. In such cases, optimization has to be performed again. The loop around the optimization ensures its convergence and may then require less examples to build the target grammar than SGL_{LS}.

Algorithm 3: ReGLiS (Learning **R**educed **G**rammar by k, l-**L**ocal **S**ubstitutability)

Input: Set of sequences K on alphabet Σ, int k, int l
Output: Grammar $G = \langle \Sigma, N_K, S_K, P_K \rangle$

```
/* Partition Sub(K) in substitutability classes */
1  C_K ← Local_substitutability_classes(K, k, l)
   /* Non-terminals for K-primes and their productions in Sub(K) */
2  N_K ← ∅, P_K ← ∅
3  for C ∈ C_K do
       /* Start symbol */
4      if C ∩ K ≠ ∅ then
5          N_K ← N_K ∪ {[[C]]}
6          S_K ← [[C]]
7          for y ∈ C do
8              P_K ← P_K ∪ {[[C]] → y}
       /* Non-terminal symbols for K-prime substitutability classes */
9      else if (|C| > 1) and (∄C' ∈ C_K : ∀y ∈ C, ∃y' ∈ C', ∃v ∈ Σ+, y = y'v) and
       (∄C' ∈ C_K : ∀y ∈ C, ∃y' ∈ C', ∃u ∈ Σ+, y = uy') then
10         N_K ← N_K ∪ {[[C]]}
11         for y ∈ C do
12             P_K ← P_K ∪ {[[C]] → y}
   /* Generalization */
13 repeat
14     P ← P_K ; P_K ← ∅
       /* Branching rules */
15     for ([[C]] → α) ∈ P ordered by increasing |α| do
16         PG ← Build_parsing_graph(α, P)
17         for β ∈ Non_redundant_rhs(PG) do
18             if ∃([[C']] → β) ∈ P_k, [[C']] ≠ [[C]] then
19                 Unify([[C']], [[C]], P_K)
20             P_K ← P_K ∪ ([[C]] → β)
21 until P_K = P
22 return ⟨Σ, N_K, S_K, P_K⟩
```

3.3. Complexity of ReGLiS algorithm

Let K be a training sample of size $|K|$ and n be the size of its longest sequence.

The complexity of the algorithm 3 is determined by three critical parts:

line 3 The creation of the local substitutability classes, which depends on the number of substrings in $Sub(K)$. This number is a main parameter for the rest of the algorithm since all operations are working on the elements of $Sub(K)$.

Algorithm 4: *Build_parsing_graph*

Input: Sequence α, Set of rules P
Output: Parsing graph $\langle V, E\rangle$

```
1  V ← {i ∈ [0, |α|]} /* vertices */
2  E ← ∅ /* labeled directed edges */
3  for i ∈ V do
4      for j ∈ V : i < j and (i, j) ≠ (0, |α|) do
5          if ∃([[C]] → α[i + 1, j]) ∈ P then
6              E ← E ∪ (i, j, [[C]])
7  return ⟨V, E⟩
```

Algorithm 5: *Non_redundant_rhs*

Input: Parsing graph: $\langle V, E\rangle$
Output: Set of non-redundant right-hand side from parsing graph: R

```
1  R ← ∅
2  paths[0] ← {{0}}
3  for i ← 1 to |V| do
       /* memorize the set of irreducible paths arriving in i */
4      P ← ⋃_{(j,i,l)∈E} (paths[j].i)
5      paths[i] ← {x ∈ P : ∄y ∈ P, y ≺_r x}
6  for path ∈ paths[n] do
7      rhs ← λ
8      for i ← 1 to |path| do
9          rhs ← rhs.β_i with β_i : (path[i − 1], path[i], β_i) ∈ E
10     R ← R ∪ rhs
11 return R
```

line 9 The reduction of the number of classes, which reduces the number of substrings to be considered in the next part of the algorithm.

line 16 The minimization of the grammar, which includes the creation of parsing graphs as the main source of complexity. It is easy to see that the complexity of this part depends on the number of rules in the grammar.

Building local substitutability classes Complexity of *Local_substitutability_classes* depends on the number of elements in $Sub(K)$. Indeed, the algorithm starts building the substitutability graph (the graph of shared contexts) by creating a vertex for each prefix of the suffix at each position of each sequence (i.e. creating the set of substrings $Sub(K)$). During this loop, a table T giving for each context its set of substrings is created.

For a sequence of length m, the maximal number of substrings created in $\mathcal{C}_K$ and in T is $m(m+1)/2$. Since n is the size of the longest sequence in K, the total number of substring occurrences is $\mathcal{O}(|K|n^2)$.

The goal then is to find the maximally connected components of the substitutability graph. By definition, each entry in table T points to a connected component. Instead of creating the whole substitutability graph, it is sufficient to create a chain for all set of

substrings in table T. The resulting graph contains the same connected components than the whole graph and the complexity remains in $\mathcal{O}(|K|n^2)$.

Note that the class $C_K(x)$ of each substring x can be stored during the search of connected components.

Reduction of the number of classes In the loop line 3–12, each element (connected component) of $\mathcal{C}_K$ is visited once, so the interior loops are performed $\mathcal{O}(|K|n^2)$ times.

Globally, the line 9 visits all substrings once ($\mathcal{O}(|K|n^2)$), and for each substring, all its prefixes and suffixes ($\mathcal{O}(2n)$). Each substring is visited only once globally because it appears in only one connected component. The treatment of each substring in each class involves an intersection of all its prefixes with other prefixes of other substrings in the same class, an operation linear with respect to the number of prefixes, i.e. $\mathcal{O}(n)$. So the global complexity of this loop is $\mathcal{O}(n^3)$.

This loop reduces in practice the number of classes and the number of substrings that belong to a substitutability class. The new set of available substrings is denoted $K-Prime$.

Generalization and minimization of the grammar In this part, (line 13 to 21), the main loop is repeated until no more right-hand side α of the grammar rules can be reduced. The worst case for this loop occurs when all substrings of length $\leq l$, $l = 1 \dots n$ are considered in turn for the reduction of α. Thus, the maximal number of steps for the loop is n.

The complexity of the interior loop (line 15 to 20) is bounded by the number of right-hand sides of the grammar $\mathcal{A} = \{\alpha \mid X \to \alpha \in P_K\}$ and their length since the loop considers all α substrings for the parsing graphs.

If t denotes the size of the target grammar (number of rules $\times$ size of rules), then the worst case complexity of generalization and minimization is $\mathcal{O}(n.t)$.

The overall *complexity of the algorithm* is then $\mathcal{O}(\max(\mathbf{n^3}, \mathbf{n.t}))$.

The rest of the paragraph further investigates this complexity to remove the dependence to the size t of the target grammar in the formula.

Initially, the number of rules is bounded by $|K - Prime|$ and the length of α is $\mathcal{O}(n)$. So the first step is bounded in $\mathcal{O}(|K - Prime|n)$. The number of rules can increase at each step of the loop, but if so, the length of the new rules decreases.

If no substring is removed during the first step of the algorithm ($|K - Prime| = |Sub(K)|$), the worst case is to get $|\mathcal{A}| = 2^{n-1}$ rules in the target grammar (Chomsky Normal Form). It occurs when each rule can be split into 2 parts at each position of α, giving rise to the maximum number of irreducible paths in the parsing graph.

If some substrings are removed, the number of irreducible paths can only decrease. For each sequence s_i of K, there exists a decomposition $s_i = u_1 v_1 u_2 v_2 \dots u_p$ such that each u_j, $j \in [1, p]$, is a substring belonging to a prime class (prime substring), each v_j, $j \in [1, p-1]$, is a substring that is not prime and $m_i = |v_1| + |v_2| + \dots + |v_{p-1}|$ is maximum.

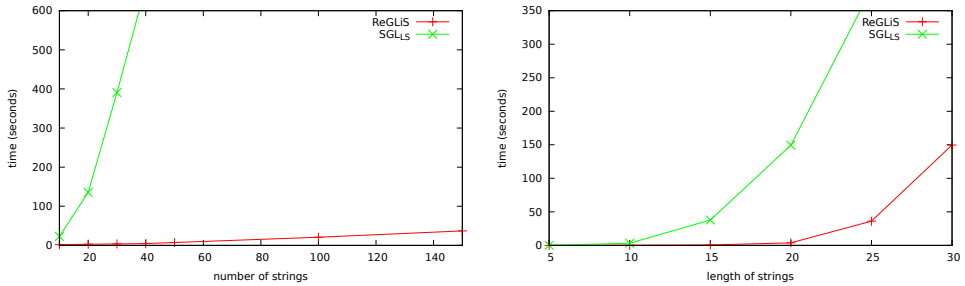
The number of decompositions of s_i is the product of the number of decompositions of primes since non-prime substrings have a single decomposition. At worst this leads to 2^{n-m_i} possibilities. For the whole training sample, the complexity of the generalization and minimization of the grammar, and thus of the whole algorithm, can thus reach $\mathcal{O}(2^{n-\min_i(m_i)})$, being dominated by the theoretical bound for the target grammar size t.

4. Experiments

4.1. Comparison of run times on simulated data

We have first launched some tests on simulated data sets to compare the practical complexity of the new algorithm introduced in this paper with respect to the algorithm used in Coste et al. (2012). The run time gain has been estimated on training samples with increasing number and increasing length of strings which were randomly generated to resemble those encountered for the protein classification task in Coste et al. (2012). For the first experiment, a random string of length 20 on a 40 symbols alphabet was randomly generated and sorted according to an arbitrary order on the alphabet to group sequentially identical characters as if it was the result of recoding a protein sequence with respect to blocks of a partial multiple sequence alignment (Coste et al., 2012; Kerbellec, 2008). To obtain a target number of similar strings, new strings were iteratively generated by replacing each character of the last generated string with a probability of 25% into a random symbol of the alphabet and by sorting the string. To study the importance of the length, the same generation procedure was used but with the number of strings set to 20 and the alphabet size set to twice the length of the sequences in the sample, this latter value coming from observation of practical protein experiments.

Results presented in figure 2 show that there is an obvious advantage in time when the number of strings increase. The length parameter has a high impact on the run time as it was expected in the complexity analysis. The curve shows that for a given amount of resource, the new algorithm allows a shift of about 10 in the length of strings, a difference that may be of high importance in practice.

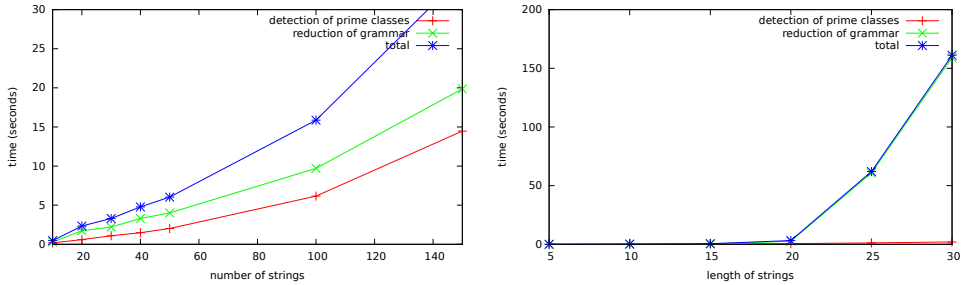


(*a*) function of the number of strings in the training sample

(*b*) function of the length of strings in the training sample

Figure 2: Run time comparison between old and new learning algorithms implementations

In figure 3, run times for the two main parts of the algorithm, the detection of prime classes and the reduction of the grammar, have been extracted from the total. As expected, the complexity is determined by the reduction part and grows exponentially in function of the length of strings in the training set.The detection of prime classes seems to stay linear, an empirical behavior better than the theoretical worst case complexity.



(a) function of the number of strings in the training sample

(b) function of the length of strings in the training sample

Figure 3: Run time for K-prime classes detection, grammar reduction and whole algorithm

4.2. Application to protein families

Motivated by the characterization of functional families of Algae enzymes (Coste et al., 2014), we aim at learning characteristic context-free grammar signatures of protein families which capture long distance sequential dependencies.

In Coste et al. (2012), we show the interest of using local substitutability on a small protein dataset extracted from Dyrka and Nebel (2009). With our new learning algorithm, it becomes then possible to run more demanding tests. By way of illustration, when our implementation of SGL_{LS} for Coste et al. (2012) required almost a day, our Python implementation of ReGLiS takes only a few minutes for one experiment. We have then been able to complete our experimentation on the whole dataset from Dyrka and Nebel (2009) and we present the results in Table 1. [1]

In all these experiments, we use a 10-fold cross validation and choose relatively small k and l parameters since learning occurs on block strings that are an order of magnitude shorter than amino acids sequences. A reasonable assumption is that some a priori knowledge may exist on the length of relevant contexts in the target application. In the case of proteins, small contexts are expected since interactions concerns few amino-acids. Practical values range from 3 to 7. In all cases, one should avoid higher values since it comes down to using (global) substitutability. Note that we have not made extensive tests on this issue and it is certainly a valuable research track.

Overall, it can be observed that local substitutability, using well chosen k and l values, significantly improves the recall with respect to global substitutability, without losing precision. Stochastic grammars get better results in terms of F-measure, except if precision is fixed to a high level. In such a case the recall obtained by ReGLiS is usually better. It is desirable to obtain a high specificity because biological experiments on proteins are expensive and a limited set of candidates can be evaluated and validated in practice.

When possible, we also compared our results with results obtained using Prosite regular expressions built on the basis of expert-provided multiple alignments of subsequences of these families. The Prosite patterns span generally around 10 positions whereas the whole protein is about 300 letters long. In consequence, such a pattern has a good recall but weak

1. Details of experiments and results are available at http://www.irisa.fr/dyliss/reglis

precision due to its generality. In contrast, our method takes into account whole sequences. The corresponding grammar has thus a high specificity. The interesting point is that despite their specificity, the level of recall obtained by the grammars appears to remain relatively high, our best recall results being comparable to those of the Prosite's patterns built by experts.

Hence, although the method can still be refined by grammar weighting schemes, this study confirms that local substitutability can be considered as a promising and effective criterion for sequence generalization, especially for the cases requiring an excellent precision.

	Zinc finger			MPI phos.		
	Precision	Recall	F-measure	Precision	Recall	F-measure
Substitutable	1	0.1	0.36	1	0.15	0.26
3,3-Local substitutable	1	0.2	0.33	1	0.5	0.67
4,4-Local substitutable	1	0.25	0.4	1	0.6	0.75
5,5-Local substitutable	1	0.33	0.5	**1**	**0.67**	**0.8**
6,6-Local substitutable	1	0.5	0.67	1	0.62	0.77
7,7-Local substitutable	**1**	**0.55**	**0.7**	1	0.53	0.69
Stochastic CFG	1	0.1	0.18	1	0.3	0.46
Dyrka and Nebel	0.15	1	0.26	0.5	1	0.67
(2009)	0.75	0.87	0.85	0.98	0.89	0.93
	PS00219			PS00063		
	Precision	Recall	F-measure	Precision	Recall	F-measure
Substitutable	1	0.2	0.33	1	0.23	0.37
3,3-Local substitutable	**1**	**0.72**	**0.84**	1	0.58	0.73
4,4-Local substitutable	1	0.7	0.82	1	0.6	0.75
5,5-Local substitutable	1	0.68	0.8	1	0.66	0.8
6,6-Local substitutable	1	0.6	0.75	**1**	**0.7**	**0.82**
7,7-Local substitutable	1	0.5	0.67	1	0.65	0.79
Prosite	1	0.6	0.75	1	0.8	0.89
Stochastic CFG	-	-	-	1	0.05	0.1
Dyrka and Nebel	-	-	-	0.1	1	0.18
(2009)	1	1	1	0.79	0.65	0.71

Table 1: Sequence class prediction by grammars obtained for different families with 10-fold cross-validation

Conclusion

With ReGLiS we have proposed another step, after the introduction of local substitutable languages, towards practical applications of substitutable language learning. To continue in this direction, on the theoretical side, we would like to better characterize the learnability gain provided by ReGLiS compared to *SGL* and, on the practical side, we would like to better understand the scope of substitutability criteria for applications and derive data-driven heuristic choices of substitutability classes.

Acknowledgments

This work benefited from the support of the French Government run by the National Research Agency and with regards to the investment expenditure programme IDEALG ANR-10-BTBR-04. GG is funded by French 'Region Bretagne' grant ARED ENZYME No. 6958.

References

D. Angluin. Inference of reversible languages. *J. ACM*, 29(3):741–765, 1982.

R. Carrascosa, F. Coste, M. Gallé, and G. G. Infante López. The smallest grammar problem as constituents choice and minimal grammar parsing. *Algorithms*, 4(4):262–284, 2011.

A. Clark. A language theoretic approach to syntactic structure. In M. Kanazawa, A. Kornai, M. Kracht, and H. Seki, editors, *The Mathematics of Language*, volume 6878 of *Lecture Notes in Computer Science*, pages 39–56. Springer Berlin Heidelberg, 2011.

A. Clark. Learning trees from strings: A strong learning algorithm for some context-free grammars. *Journal of Machine Learning Research*, 14:3537–3559, 2014.

A. Clark and R. Eyraud. Polynomial identification in the limit of substitutable context-free languages. *Journal of Machine Learning Research*, 8:1725–1745, August 2007.

F. Coste, G. Garet, and J. Nicolas. Local Substitutability for Sequence Generalization. In Jeffrey Heinz, Colin de la Higuera, and Tim Oates, editors, *ICGI 2012*, volume 21 of *JMLR Workshop and Conference Proceedings*, pages 97–111, Washington, États-Unis, Sep 2012. MIT Press.

F. Coste, G. Garet, A. Groisillier, J. Nicolas, and T. Tonon. Automated enzyme classification by formal concept analysis. In C. Vera Glodeanu, M. Kaytoue, and C. Sacarea, editors, *ICFCA*, volume 8478 of *Lecture Notes in Computer Science*, pages 235–250. Springer, 2014. ISBN 978-3-319-07247-0.

W. Dyrka and J.-C. Nebel. A stochastic context free grammar based framework for analysis of protein sequences. *BMC Bioinformatics*, 10(1):323+, October 2009.

G. Kerbellec. *Apprentissage d'automates modélisant des familles de séquences protéiques.* PhD thesis, Université Rennes 1, 2008.

R. Yoshinaka. Identification in the limit of (k,l)-substitutable context-free languages. In *Proceedings of the 9th international colloquium conference on Grammatical inference: theoretical results and applications*, ICGI'08, pages 266–279, 2008.

Appendix A. Example of reduction of grammar on natural language

In this appendix, we illustrate the interest of the prime classes and the reduced grammar form on the following training set K:

$$\begin{aligned}\mathbf{K} = \{&\text{"Major General was here yesterday morning."},\\ &\text{"Major General went here yesterday morning."},\\ &\text{"Major General will be there tomorrow morning."},\\ &\text{"He will be gone tomorrow evening."}\}\end{aligned}$$

Then the grammar obtained by the classical approach introducing a non-terminal for each substitutability class and splitting the rules in all possible rules in Chomsky normal form would be:

$X_{47} \to' yesterday'$
$X_{46} \to X_3X_{43}|X_{11}X_{19}|X_{23}X_{13}$
$X_{45} \to X_{20}X_2$
$X_{44} \to X_{20}X_1|X_{45}X_{29}|X_{34}X_{16}|X_9X_{15}$
$X_{43} \to X_{20}X_{19}|X_{45}X_{13}$
$X_{42} \to' tomorrow'$
$X_{41} \to X_{42}X_{15}$
$X_{40} \to X_{30}X_{46}|X_{21}X_{43}|X_{39}X_{19}|X_{25}X_{13}|X_{21}X_{34}|X_8X_{13}$
$S \to X_{30}X_{24}|X_{21}X_{31}|X_{10}X_{32}|X_{36}X_{17}|X_{26}X_{15}|X_{39}X_1$
$|X_{25}X_{29}|X_{40}X_{41}|X_{21}X_{44}|X_8X_{29}|X_{40}X_{16}|X_{33}X_{15}$
$X_{29} \to X_{13}X_{16}|X_4X_{15}|X_{13}X_{41}|X_{38}X_{15}$
$X_{28} \to X_{27}X_{47}$
$X_{25} \to X_{30}X_{23}|X_{21}X_{45}|X_{39}X_2$
$X_{24} \to X_3X_{31}|X_{18}X_{32}|X_{37}X_{17}|X_{14}X_{15}|X_{11}X_1|X_{23}X_{29}|X_{46}X_{41}$
$X_{27} \to' here'$
$X_{26} \to X_{30}X_{14}|X_{21}X_{12}|X_{10}X_{28}|X_{36}X_{47}|X_{39}X_{35}|X_{25}X_{38}|X_{40}X_{42}$
$X_{21} \to' He'|X_{30}X_3$
$X_{20} \to' will'$
$X_{23} \to X_3X_{45}|X_{11}X_2$
$X_{22} \to X_6X_{27}$
$X_8 \to X_{21}X_{45}|X_{39}X_2$
$X_9 \to X_{20}X_5|X_{45}X_4|X_{34}X_{42}$
$X_2 \to' be'$
$X_3 \to' Major'$
$X_1 \to X_2X_{29}|X_{19}X_{16}|X_5X_{15}|X_{19}X_{41}|X_{35}X_{15}$
$X_6 \to' was'|'went'$
$X_4 \to X_{13}X_{42}$
$X_5 \to X_2X_4|X_{19}X_{42}$
$X_{32} \to X_{27}X_{17}|X_{28}X_{15}$
$X_{33} \to X_{21}X_9|X_{39}X_5|X_8X_4|X_{40}X_{42}$
$X_{30} \to General$
$X_{31} \to X_6X_{32}|X_{22}X_{17}|X_{12}X_{15}|X_{20}X_1|X_{45}X_{29}|X_{43}X_{41}$
$X_{36} \to X_{30}X_{37}|X_{21}X_{22}|X_{10}X_{27}$
$X_{37} \to X_3X_{22}|X_{18}X_{27}$
$X_{34} \to X_{20}X_{19}|X_{45}X_{13}$
$X_{35} \to X_2X_{38}|X_{19}X_{42}$
$X_{38} \to X_{13}X_{42}$
$X_{39} \to X_{30}X_{11}|X_{21}X_{20}$
$X_{18} \to X_3X_6$
$X_{19} \to X_2X_{13}$
$X_{10} \to X_{30}X_{18}|X_{21}X_6$
$X_{11} \to X_3X_{20}$
$X_{12} \to X_6X_{28}|X_{22}X_{47}|X_{20}X_{35}|X_{45}X_{38}|X_{43}X_{42}$
$X_{13} \to' there'|'gone'$
$X_{14} \to X_3X_{12}|X_{18}X_{28}|X_{37}X_{47}|X_{11}X_{35}|X_{23}X_{38}|X_{46}X_{42}$
$X_{15} \to' morning'|'evening'$
$X_{16} \to X_{42}X_{15}$
$X_{17} \to X_{47}X_{15}$

while the **reduced grammar** on K-prime classes, would simply be:

$$S \to X_3X_4X_2$$
$$X_1 \to was \mid went$$
$$X_2 \to morning \mid evening$$
$$X_3 \to He \mid Major\ General$$
$$X_4 \to will\ be\ X_5\ tomorrow \mid X_1\ here\ yesterday$$
$$X_5 \to there \mid gone$$