Discovering grammar of an unknown text as an optimisation problem

P.A. RYSZAWA
pawel.ryszawa@wat.edu.pl

Military University of Technology, Faculty of Cybernetics
Urbanowicza Str. 2, 00-908 Warsaw, Poland

This paper concerns the problem of discovering an unknown grammar from a text sample. The discovering methods are formulated as optimisation problems based on a binary representation of context-sensitive grammars. The representation starts with a longest possible vector of bits to, finally, make it more compact so as to be usable in practical applications. For the sake of simplicity, considered are only noncontracting (length-preserving) grammars of order 2, excluding productions of the form $P: A \to B$ and those deriving the empty string, i.e $P: A \to \varepsilon$.

Keywords: context-sensitive language, formal grammar, noncontracting grammar, binary representation, optimisation problem.

1. Introduction

The notion of the formal grammar needs some representation, if we want to use it in any algorithm run on a computer. Here, in this article, the objective is to develop a binary representation, easy to use and to formulate an optimisation problem. A binary representation $\mathcal{B}(G)$ of a grammar G is considered to be a string of the form:

$$\mathcal{B}(G) = [b_1, b_2, \dots, b_\Xi], \tag{1}$$

where Ξ is its (fixed) length, $b_i \in \{0,1\}$, and $i = 1,2, \dots, \Xi$.

It is assumed throughout this paper that each production P_i in G is of one of the following forms:

$$P_i: A \to BC, \tag{2}$$

$$P_i: AB \to CD, \tag{3}$$

where $A, B, C, D \in V_N$ are nonterminal symbols, or

$$P_i: A \to a, \tag{4}$$

where $A \in V_N$ is a nonterminal symbol and $a \in V_T$ is a terminal symbol. The productions of the form $P_i: A \to B$ are ruled out as redundant. The forms (2), (3) and (4) are sufficient to represent a noncontracting grammar of order 2 – a type of grammar weakly equivalent to noncontracting grammars of any order, and hence weakly equivalent to any context-sensitive grammar (see [3]).

It is required, of course, that each grammar G must have its own unique binary representation $\mathcal{B}(G)$. As a rule of thumb, two similar grammars should also have similar representations. "Similar" means here that some metric could be defined, in which two short-distant grammars have two short-distant corresponding representations. Likewise, the binary representations of two long-distant grammars should also be long-distant.

Finally, it is also assumed that there exists a space of grammars. Since it is needed for formulating an optimisation problem on this space, the number of possible nonterminal and terminal symbols, as well as the number of possible productions, is limited and considered constant. Thus, the space $\mathcal{S} = \mathcal{S}(\widetilde{\boldsymbol{V}}_N, \widetilde{\boldsymbol{V}}_T)$ of grammars of the form $G = \langle V_N, V_T, P, S \rangle$ could be defined as $\mathcal{S} = \{G \mid V_N \subseteq \widetilde{\boldsymbol{V}}_N \wedge V_T \subseteq \widetilde{\boldsymbol{V}}_T\}$. $\widetilde{\boldsymbol{V}}_N$ is the maximal set of nonterminal nodes, i.e. every nonterminal symbol of a grammar $G \in \mathcal{S}(\widetilde{\boldsymbol{V}}_N, \widetilde{\boldsymbol{V}}_T)$ belongs to $\widetilde{\boldsymbol{V}}_N$. Likewise, $\widetilde{\boldsymbol{V}}_T$ is the maximal set of terminal nodes, i.e. every terminal symbol of a grammar $G \in \mathcal{S}(\widetilde{\boldsymbol{V}}_N, \widetilde{\boldsymbol{V}}_T)$ belongs to $\widetilde{\boldsymbol{V}}_T$. Since adding new elements to V_N or V_T, but without adding any production to P, does not change the grammar G, i.e. both are still weakly equivalent, the space $\mathcal{S}$ of grammars can also be defined as

$$\mathcal{S}(\widetilde{\boldsymbol{V}}_N, \widetilde{\boldsymbol{V}}_T) = \{G \mid V_N = \widetilde{\boldsymbol{V}}_N \wedge V_T = \widetilde{\boldsymbol{V}}_T\}. \tag{5}$$

That is, each $G = \langle V_N, V_T, P, S \rangle \in \mathcal{S}$ can be extended to $G = \langle \widetilde{\boldsymbol{V}}_N, \widetilde{\boldsymbol{V}}_T, P, S \rangle$ and still not

affecting the language generated by G. From now on, the latter definition is used in this paper.

2. General concept of binary representations of grammars

Assume we have a universal set $\widetilde{\boldsymbol{V}}_N$ of nonterminals, a universal set $\widetilde{\boldsymbol{V}}_T$ of terminals, and a universal set $\widetilde{\boldsymbol{P}}$ of production rules. The maximal space of grammars, that we can define in more detail now, is

$$\mathcal{S} = \{G \mid V_N = \widetilde{\boldsymbol{V}}_N \wedge V_T = \widetilde{\boldsymbol{V}}_T \wedge P \subseteq \widetilde{\boldsymbol{P}}\}. \quad (6)$$

Hence, two distinct grammars in the space $\mathcal{S}$ are distinguishable only by their subsets of production rules.

If enumerating all the possible production rules from $\widetilde{\boldsymbol{P}}$, i.e. mapping each production rule to its unique index, all the grammars in $\mathcal{S}$ would be encoded with a binary string as follows: Let $\vec{\boldsymbol{P}} = [P_1, P_2, \dots, P_\Xi]$ be an ordered vector of all possible productions, indexed from 1 through Ξ, where, of course, $\Xi = |\widetilde{\boldsymbol{P}}|$. Now, as per the definition (1), let $\mathcal{B}(G) = [b_1, b_2, \dots, b_\Xi]$ be such that:

$$b_i = \begin{cases} 0, & P_i \notin P, \\ 1, & P_i \in P, \end{cases} \quad (7)$$

where P is the set of production rules of G.

The general concept of binary representations depends on how $\mathcal{S}$ is defined. This, in turn, depends on assumed sets of possible terminals $\widetilde{\boldsymbol{V}}_T$ and nonterminals $\widetilde{\boldsymbol{V}}_N$ along with the form of possible production rules. As soon as the above sets are determined, the order of $\vec{\boldsymbol{P}}$ must also be determined. Thus, the binary representation is well defined. It is also obvious that two grammar differing by only one production rule have two representations differing by only one bit.

3. Simple binary representation of noncontracting grammars of order 2

Assume that for all grammars G in $\mathcal{S}$ the set of terminals $\widetilde{\boldsymbol{V}}_T$ contains as many elements as $\widetilde{\boldsymbol{V}}_N$ and for each $\mu \in \widetilde{\boldsymbol{V}}_N$ there exists exactly one $\gamma \in \widetilde{\boldsymbol{V}}_T$ and exactly one production of the form (4), that is $\mu \to \gamma$, that belongs to $\widetilde{\boldsymbol{P}}$. Given $\widetilde{\boldsymbol{V}}_N$ and $\widetilde{\boldsymbol{P}}$ let us now define the order $\vec{\boldsymbol{P}}$. Assume that first group of bits in the constructed representation is associated with productions of the form (2). Next, those of the form (3). The productions of the form (4) can be omitted in the binary representation as these are always the same, independently of the underlying grammar G.

First, note that the productions of the form (2) and (3) have 3 or 4 nonterminal symbols, respectively. Hence, there can be Υ^3 or Υ^4 such possible productions, respectively, where $\Upsilon = |\widetilde{\boldsymbol{V}}_N| = |\widetilde{\boldsymbol{V}}_T|$. Thus, the overall structure of the binary representation can be split into 2 groups as follows:

$$\mathcal{B}(G) = [b_1, \dots, b_{\Upsilon^3}, b_{\Upsilon^3+1}, \dots, b_{\Upsilon^3+\Upsilon^4}] \quad (8)$$

Both of these groups of Υ^3 and Υ^4 elements, respectively, correspond to the production rules of the form (2) and (3). The order within each group is also of the concern.

Assume we have an ordered vector of terminal symbols $\vec{\boldsymbol{V}}_T = [\mu_1, \mu_2, \dots, \mu_\Upsilon]$ and the corresponding ordered vector of nonterminal symbols $\vec{\boldsymbol{V}}_N = [\gamma_1, \gamma_2, \dots, \gamma_\Upsilon]$, such that $\gamma_i \to \mu_i$ are productions of the form (4) assumed to belong to $\widetilde{\boldsymbol{P}}$, where $i = 1,2, \dots, \Upsilon$. Let us also index the symbols in a production from left to right. Then, all the production of the same form can be ordered lexicographically. For example, the group of productions of the form (3) is ordered as follows: $\gamma_1 \to \gamma_1\gamma_1$, $\gamma_2 \to \gamma_1\gamma_1$, ..., $\gamma_\Upsilon \to \gamma_1\gamma_1, \gamma_1 \to \gamma_2\gamma_1$, ... , $\gamma_\Upsilon \to \gamma_\Upsilon\gamma_\Upsilon$, and the productions of the form (4) – from $\gamma_1\gamma_1 \to \gamma_1\gamma_1, \gamma_2\gamma_1 \to \gamma_1\gamma_1$, and so on... through to $\gamma_\Upsilon\gamma_\Upsilon \to \gamma_\Upsilon\gamma_\Upsilon$.

4. Between bit indices and production rules

For practical applications, it would be essential to quickly identify a particular bit index in the binary vector with the production it represents. On the other hand, the opposite task of quickly finding the bit representing a particular production rule would also be of importance.

Given a bit index `i` and $\Upsilon = |\widetilde{\boldsymbol{V}}_N|$, the first step is to determine the form of the corresponding production. This is done in the following manner:

```
if i <= Υ^3 then
  prod_len := 3;
else if i <= Υ^3 + Υ^4 then
  prod_len := 4;
  i := i - Υ^3;
end if;
```

Having determined the form of the production rule (variable `prod_len`, saying how many symbols it has), the rightmost, the second rightmost nonterminal symbols, through to the leftmost one, can be found:

```
x := prod_len;
while x > 0 do
  s[x] := (i - 1) mod ϒ + 1;
  i := (i - 1) div ϒ + 1;
  x := x - 1;
end while;
```

The table `s`, indexed from 1 to p, where p is the number of nonterminal symbols in the particular form of production rule, contains the result. `s[1]` contains the leftmost nonterminal symbol number, i.e. `s[1]`$= a$ means it is γ_a. Similarly, `s[`p`]` contains the number of the rightmost nonterminal symbol.

To convert oppositely, i.e. to determine the corresponding bit index, given a production rule, one needs to first set `prod_len` variable to the number of symbols in the rule and set table `s` elements to numbers representing the nonterminal symbols. The interpretation of `prod_len` and `s` is the same as previously. Having set them, the following algorithm gives the bit index in the result variable `i`:

```
i := 0;
x := 1;
while x <= prod_len do
  i := i + s[x] * ϒ ^ (x - 1);
  if x > 2 then
    i := i + ϒ ^ (x - 1);
  end if;
  x := x + 1;
end while;
```

5. Optimisation problems for discrimination

Assume there is a formal language L the grammar of which is unknown. The aim is to find a feasible grammar that best describes it. Assume also that the sentences belonging to this language are known. It means that the resulting grammar should accept these sentences while rejecting all the others. The overall optimisation problem can be formulated as follows: Find a grammar $G_0 = \mathcal{B}^{-1}(\boldsymbol{b}_0)$, represented by a binary string $\boldsymbol{b}_0 \in D$, accepting all $\mathfrak{T} \in \widetilde{\boldsymbol{V}}_T^+$, if $\mathfrak{T} \in L$. However, G_0 should reject all $\mathfrak{U} \in \widetilde{\boldsymbol{V}}_T^+$, if $\mathfrak{U} \notin L$. That is,

$$\begin{aligned} \forall_{\mathfrak{T}\in L}\, f(\boldsymbol{b}_0, \mathfrak{T}) &= \max\nolimits_{\boldsymbol{b}\in D}\{f(\boldsymbol{b}, \mathfrak{T})\}, \\ \forall_{\mathfrak{U}\notin L}\, f(\boldsymbol{b}_0, \mathfrak{U}) &= \min\nolimits_{\boldsymbol{b}\in D}\{f(\boldsymbol{b}, \mathfrak{U})\}, \end{aligned} \qquad (9)$$

where:
$f: D \times \widetilde{\boldsymbol{V}}_T^+ \to \mathbb{R}$ – some **fitness function** that evaluates the grammar represented by $\boldsymbol{b} \in D$ against a given sentence from $\widetilde{\boldsymbol{V}}_T^+$,
$D = \{0,1\}^{\Xi}$ – is the domain (the set of all possible grammar representations),
$\Xi = \Upsilon^3 + \Upsilon^4$ – the number of bits necessary to represent a grammar.

It is not necessary to find "ideal" G_0. Perhaps, for some practical application, it would be enough to find one highly fitted to $\mathfrak{T}$, whereas sufficiently discriminating from any $\mathfrak{U} \neq \mathfrak{T}$. Thus, it slightly modifies the problem (9) to:

$$\begin{aligned} &\forall_{\mathfrak{T}\in L}\, f(\boldsymbol{b}, \mathfrak{T}) \to \max, \\ &\forall_{\mathfrak{U}\notin L}\, f(\boldsymbol{b}, \mathfrak{U}) \to \min, \\ &\quad \text{for } \mathfrak{U} \neq \mathfrak{T}. \end{aligned} \qquad (10)$$

That is, the problem is not strictly aimed at finding the correct grammar but a "good enough" one. Some properties of a language subject to such routine might be revealed, perhaps, based on the structure of the grammar found – not necessarily a perfect one.

It is assumed in problem (9) and (10) that f can take values from some continuum domain. In the simplest problem, though, f usually takes either 0, if a sentence is not recognized against a given grammar, or 1, if it is recognized indeed.

6. Natural language properties

Assume that not all the sentences of a language are known, but having known some of them all the others can be deduced in some way. Assume it can be deducted in the following way:

Proposition 1
Let $\upsilon = \chi_1\chi_2 \dots \chi_n$ be a known sentence in language L, where each χ_i is a subsentence consisting of a number of terminal symbols, i.e. $\chi_i \in \widetilde{\boldsymbol{V}}_T^+$. Let also $\sigma: \{1, \dots, n\} \to \{1, \dots, n\}$ be an n-element permutation. It is postulated that another sentence $\upsilon' = \chi_{\sigma(1)}\chi_{\sigma(2)} \cdots \chi_{\sigma(n)}$, hashing subsentences of υ, belongs to L with high probability.

This is not necessary that $\upsilon' \in L$. Though, it is believed to be highly probable. The more chunks the sentence is split into and the more hashed they are, the less probable is that the resulting hashed sentence belongs to L. Let us define the measure, denoted by $\mathcal{Z}(\sigma)$, of how "strong"

is a particular permutation σ, i.e. how much it hashes v. Then,

$$\mathcal{Z}(\sigma) \triangleq \triangleq \left| \left\{ \langle \sigma(s_1), \sigma(s_2) \rangle \middle| \begin{matrix} \sigma(s_2) \neq \sigma(s_1) + 1, \\ s_2 = s_1 + 1, \\ s_1, s_2 = 1, 2, \dots, |\mathcal{N}| + 1 \end{matrix} \right\} \right|. \quad (11)$$

Proposition 2
Let v, v' and σ be as in proposition 1. It is postulated that the probability $\Pr\{v' \in L\}$ is somehow inversely proportionate to $\mathcal{Z}(\sigma)$.

The above propositions are based on observations of the natural languages. The sentence, in the sense of theory of languages, can be the text of some book, an article or just a mere sentence in terms of natural languages. Its lexemes (terminal symbols) are usually natural words. Some languages allow to compose a sentence with some freedom in the word order – e.g. Polish, Spanish. Some, on the other hand, are more strict in that, however still give some freedom – e.g. English. Given a natural text, that is a book, an article, a sentence, etc., one can split it in $n - 1$ points and hash it, i.e. apply some permutation on groups of words. The permuted text perhaps still can be considered a correct one in the language, but this is less likely for highly hashed texts. Let us call it the **hashing property** of a natural language. Please note that, if we split a particular text in a considerably high number of points and hash, some indivisible parts of the text might be damaged. The more such points of cut, the more the probability of such a consequence.

Assume now, that there is given a text of an unknown origin and hence its language and grammar are unknown. It seems barely possible that another text written with the same set of words exists but not belonging to the language in question. In terms of the theory of languages, it means that it is almost impossible to have another sentence with terminal symbols picked up from the same set, whereas it is a sentence from another language. It can be heuristically assumed, though, that the hashing property holds.

Although it is not always true, the natural languages are usually described with as simple grammar rules as possible. This means, researchers intend to describe the majority of correct sentences in a given language with less rules. Let us call it the **minimal structure property**. In terms of the theory of languages, this means there should be as few production rules as possible. This property implicitly assumes that the smallest possible grammar will describe the language in question while it rejects the sentences from another language. That is, if only the grammar contained more production rules than necessary, it could accept incorrect sentences. It should be emphasized here, that this is a heuristic assumption only!

7. Optimisation problem for finding grammar

Now, let us move on to construct further optimisation problems for finding grammars. Assuming minimal structure property of the language, a new requirement on the number of productions is imposed on the optimisation problem. Thus, the problem (10) becomes to:

$$\begin{aligned} &\forall_{\mathfrak{T} \in L} f(\boldsymbol{b}, \mathfrak{T}) \to \max, \\ &\forall_{\mathfrak{U} \notin L} f(\boldsymbol{b}, \mathfrak{U}) \to \min, \\ &\qquad \text{for } \mathfrak{U} \neq \mathfrak{T}. \\ &\boldsymbol{b}\mathbf{1}^T \to \min, \end{aligned} \quad (12)$$

where $\mathbf{1} = [1, 1, \dots, 1]$.

Assuming the hashing property, further transformations of the optimisation problem can be introduced. If based on only one sentence, it is hardly possible to determine all the $\mathfrak{U} \notin L$ (while having only one $\mathfrak{T} \in L$).

Let us first define a hash function h. Of course, one of its necessary parameters is the sentence $\mathfrak{T} \in \widetilde{\boldsymbol{V}}_T^+$. The other parameters should determine the number and the locations of splitting points, and the way chunks of the base sentence are arranged in the resulting text. The latter is most easily expressed with the notion of permutation. The hash function is defined then as:

$$h: \mathcal{H} \to \widetilde{\boldsymbol{V}}_T^+ \quad (13)$$

where:
$\mathcal{H} \subseteq \widetilde{\boldsymbol{V}}_T^+ \times 2^{\mathbb{N}_+} \times \mathcal{P}$ is parameter domain,
$\mathcal{P}$ – a set of all possible permutations,
The domain $\mathcal{H}$ is such a subset of $\widetilde{\boldsymbol{V}}_T^+ \times 2^{\mathbb{N}_+} \times \mathcal{P}$ that meets the following constraints: Let $\langle \mathfrak{T}, \mathcal{N}, \sigma \rangle \in \mathcal{H}$, where $\mathcal{N}$ is a subset of positive natural numbers, σ is some permutation, then

a) the length of the text minus 1 is the maximal possible index of a splitting point as this is the number of locations between each two neighbouring terminal symbols, i.e.
$$\sup \mathcal{N} < |\mathfrak{T}| \quad (14)$$
b) the cardinality of the permutation is equal to that of splitting points plus 1, as this is

the number of chunks to be arranged around the splitting points, i.e.

$$\sigma: \{1, 2, \dots, |\mathcal{N}| + 1\} \to \{1, 2, \dots, |\mathcal{N}| + 1\} \tag{15}$$

is a bijection on $\{1, 2, \dots, |\mathcal{N}| + 1\}$.

The set of splitting point locations contains the indices of terminal-symbols in the text $\mathfrak{T} \in \widetilde{\boldsymbol{V}}_T^+$ ($\mathfrak{T}$ – passed as the very first parameter to h), after which the splitting point is located.

The more the given sentence $\mathfrak{T}$ is hashed by h, the less are the expectations that this sentence fits to the (unknown yet) grammar G_0 – as per proposition 2. The measure of the hashing level is $\mathcal{Z}(\sigma)$ – as per (11), and should proportionately affect the fitness function f when looking for extrema.

Let $\mathcal{N}$ be a fixed set of splitting locations and Ω be some set of permutations as per (15). It is not necessary that all possible permutations around the splitting points $\mathcal{N}$ be used. The way Ω should be constructed is beyond the scope of this article. Let $\mathcal{T}$ denote a set of sentences hashed from $\mathfrak{T}$ with permutations from Ω, paired with their corresponding hashing level, i.e.

$$\mathcal{T} = \left\{ \langle \mathfrak{T}', z \rangle \;\middle|\; \begin{array}{c} \mathfrak{T}' = h(\mathfrak{T}, \mathcal{N}, \sigma) \\ z = \mathcal{Z}(\sigma) \\ \sigma \in \Omega \end{array} \right\} \tag{16}$$

Assume also that some arbitrary nondecreasing monotonic function $d: S \to \mathbb{R}_+$ is given for either augmenting (if d is considerably increasing) or reducing (if d is moderately increasing) the effects of $\mathcal{Z}(\sigma)$, i.e.

$$\begin{array}{l} d: S \to \mathbb{R}_+, \\ z_1 < z_2 \Leftrightarrow\ d(z_1) \le d(z_2) \\ z_1, z_1 \in S \end{array} \tag{17}$$

where $S \subseteq \mathbb{R}$ covers at least all the possible values of $\mathcal{Z}(\sigma)$, d take positive real values.

The optimisation problem is based on (12) and is derived as follows: The fitness function is computed for all $\mathfrak{T}'$ that were generated from $\mathfrak{T}$. Since high hashing level z of a sentence $\mathfrak{T}'$, as expressed by the corresponding pair $\langle \mathfrak{T}', z \rangle \in \mathcal{T}$, is agreed to make $\mathfrak{T}'$ less important in the overall problem, this sentence should have less impact on the criterion value. Thus, the compound "max" criterion can be produced from those maximized ones in (12), by weighting them appropriately.

$$\sum_{\langle \mathfrak{T}', z \rangle \in \mathcal{T}} \frac{f(\boldsymbol{b}, \mathfrak{T}')}{d(z)} \to \max,$$

$$\boldsymbol{b}\boldsymbol{1}^T \to \min, \tag{18}$$

This is a two criteria model. The functions f and d should be arbitrarily chosen. The above formulas give only general hints on how to do it. The f should be the result of some algorithm testing the membership of $\mathfrak{T}'$ in $L(G)$, while $G = G(\boldsymbol{b})$.

Instead of minimizing criteria, one can just impose a limit on the number of productions. Thus, the problem reduces to a single criteria one subject to a limit on productions. That is,

$$\sum_{\langle \mathfrak{T}', z \rangle \in \mathcal{T}} \frac{f(\boldsymbol{b}, \mathfrak{T}')}{d(z)} \to \max,$$

subject to

$$\boldsymbol{b}\boldsymbol{1}^T \le B, \tag{19}$$

where B is the maximal allowed number of production rules.

The domain of the optimisation problem can be further constrained. Let $\mathfrak{B}$ denote the domain. Then, the problem (19) can have the general form:

$$\sum_{\langle \mathfrak{T}', z \rangle \in \mathcal{T}} \frac{f(\boldsymbol{b}, \mathfrak{T}')}{d(z)} \to \max,$$

subject to

$$\boldsymbol{b} \in \mathfrak{B}. \tag{20}$$

Of course, $\mathfrak{B}$ can incorporate the constraint as in (19), and perhaps some other.

Example 1

Let $\mathfrak{T} = abab$ be a sentence from an unknown language (generated by some unknown grammar). The terminal symbols $V_T = \{a, b\}$ are assumed, by definition, to correspond to some nonterminal symbols, e.g. $A, B \in V_N$. Thus, the productions $A \to a$ and $B \to b$ are assumed to be in every grammar considered in the problem. Assume further that the goal is to find a grammar of at most 4 nonterminals. Let these be $V_N = \{A, B, D, S\}$. Finally, assume that, apart from the aforementioned productions, all the other productions are of the form $V \to WY$, i.e. that those of the form $VW \to YZ$ are disallowed.

Since $\Upsilon = 4$ here, the number of bits to represent the potential grammar is $\Upsilon^3 + \Upsilon^4 = 4^3 + 4^4 = 64 + 256 = 320$. The 320 productions corresponding to each bit of the binary representation are: $A \to AA$, $B \to AA$, $D \to AA$, $S \to AA$, $A \to BA$, $B \to BA$, $D \to BA$, $S \to BA$,..., $S \to SS$, $AA \to AA$, $BA \to AA$, ..., $SS \to SS$. Of course the last 256 productions are disallowed, hence the domain is defined as

$$\mathfrak{B} = \{[b_1, b_2, \dots, b_{320}] \in \{0,1\}^{320} \mid b_{65} = b_{66} = \dots = b_{320} = 0\}.$$

Let the fitness function f be defined as follows: If the CYK algorithm (see [5], [11], [7]) gives a nonempty set of symbols that derive the whole sentence in a grammar $G = \mathcal{B}^{-1}(\boldsymbol{b})$, the function takes value of 1. If, on the other hand, the subsentence $\mathfrak{S}$ of the sentence $\mathfrak{T}$ is among the longest ones for which a nonterminal was found, then the fitness function value is a fraction $f(b, \mathfrak{T}) = {|\mathfrak{S}|}/{|\mathfrak{T}|}$. The motivation to define f in such a way is that, perhaps, the longer the substring that can be derived with the grammar found so far, the closer is the routine to the optimum.

Let $\boldsymbol{b}_1 \in \{0,1\}^{320}$ have 1 at the 7th, 20th and 33rd positions (0 at the other ones), i.e. $\boldsymbol{b}_1 =$ $= 00000010000000000001000000000010$ ... – it is easily seen that the grammar $G_1 = \mathcal{B}^{-1}(\boldsymbol{b}_1)$ contains the productions $D \to BA$, $S \to AB$ and $A \to AD$, which result from the above value of b explicitly. Of course, it also implicitly contains $A \to a$ and $B \to b$. The sentence $\mathfrak{T} = abab$ is directly reduced to $ABAB$ with those implicit production rules and then the CYK algorithm continues. With the grammar G_1 its results are as follows:

$X_{14} = \{S\}$			
$X_{13} = \{A\}$	$X_{24} = \emptyset$		
$X_{12} = \{S\}$	$X_{23} = \{D\}$	$X_{34} = \{S\}$	
$X_{11} = \{A\}$	$X_{22} = \{B\}$	$X_{33} = \{A\}$	$X_{44} = \{B\}$

Hence, $f(\boldsymbol{b}_1, \mathfrak{T}) = 1$, as it has been found that the whole sentence $abab$ can be derived with G_1 starting from the S.

Let now $\boldsymbol{b}_2 \in \{0,1\}^{320}$ have 1 at the 20th position and 0 elsewhere, that is $\boldsymbol{b}_2 = 00000000000000000010 \dots 0$. Then, the grammar $G_2 = \mathcal{B}^{-1}(\boldsymbol{b}_2)$ contains only the production $S \to AB$. The following table shows the results of the CYK algorithm on this grammar:

$X_{14} = \emptyset$			
$X_{13} = \emptyset$	$X_{24} = \emptyset$		
$X_{12} = \{S\}$	$X_{23} = \emptyset$	$X_{34} = \{S\}$	
$X_{11} = \{A\}$	$X_{22} = \{B\}$	$X_{33} = \{A\}$	$X_{44} = \{B\}$

The longest substrings are those represented by X_{12} and X_{34}, i.e. of length 2 Thus, $f(\boldsymbol{b}_2, \mathfrak{T}) = {2}/{4} = 0.5 < 1$.

Let also $\boldsymbol{b}_3 \in \{0,1\}^{320}$ have 1 at the 20th and 64th positions, and 0 elsewhere. The grammar $G_3 = \mathcal{B}^{-1}(\boldsymbol{b}_3)$ it represents contains two productions: $S \to AB$ and $S \to SS$. The result of the CYK algorithm is as follows:

$X_{14} = \{S\}$			
$X_{13} = \{A\}$	$X_{24} = \emptyset$		
$X_{12} = \{S\}$	$X_{23} = \emptyset$	$X_{34} = \{S\}$	
$X_{11} = \{A\}$	$X_{22} = \{B\}$	$X_{33} = \{A\}$	$X_{44} = \{B\}$

Thus, $f(\boldsymbol{b}_3, \mathfrak{T}) = 1 = f(\boldsymbol{b}_1, \mathfrak{T})$.

Example 2

Let us consider the context-sensitive parsing algorithm of Woods, as in [10]. The allowed production rule forms are: context-free $A \to BC$, left context-sensitive $AB \to AC$ and right context-sensitive $AB \to CB$. Assume the nonterminal symbols set consists of 3 items. It is easily seen that, among $\Xi = \Upsilon^3 + \Upsilon^4 = 3^3 + 3^4 = 27 + 81 = 108$ possible productions, there are $3 \cdot 3^3 = 81$ allowed. This is due to the fact that left and right context-sensitive rules have 3 degree of freedom each, as many as the context-free rules.

The domain $\mathfrak{B}$ consists of such $[b_1, b_2, \dots, b_{108}] \in \{0,1\}^{108}$ that must have $b_i = 0$ for i = 32, 33, 35, 36, 40, 42, 43, 45, 49, 50, 52, 53, 56, 57, 62, 63, 64, 66, 70, 72, 73, 74, 79, 80, 83, 84, 86, 87, 91, 93, 94, 96, 100, 101, 103, 104. Among the 108 rules $A \to AA$, $B \to AA$, ..., $C \to CC$, $AA \to AA$, $BA \to AA$, ..., $CC \to CC$, the above indices point out the "forbidden" rules. For instance, the 32nd production rule is $BB \to AA$ and has 4 symbols, but is neither left nor right context-sensitive rule.

Example 3

Let the terminals V_T consist of 4 symbols (lexemes): "el", "gato", "es" and "negro", i.e. V_T = {<el>, <gato>, <es>, <negro>}. Let us have the sentence $\mathfrak{T}$ = "el gato es negro". There are many possible permutations of $\mathfrak{T}$, for instance $\mathfrak{T}'$ ="negro es el gato" or $\mathfrak{T}''$ ="es negro el gato". It is believed that $\mathfrak{T}'$ and $\mathfrak{T}''$ form correct sentences in some language, hence

they can be used as an input to the optimisation problem of finding grammar. This is true in fact, as all of the above sentences are taken from the Spanish language and mean "the cat is black". It is only a matter of style to change the word order. However, $\mathfrak{T}''' =$ "negro gato es el" is rather incorrect in Spanish (unless "el" is replaced by "él", but then the sentence would have different meaning: "black cat is him").

Example 4
Let $\mathfrak{T}$, $V_N, \mathfrak{B}$, $\boldsymbol{b}_1$ and $\boldsymbol{b}_3$ be as in example 1. Both b_1 and b_3 give the same value of the fitting function, thus, are considered equally. However, $\boldsymbol{b}_3$ represents a grammar of 2 production rules only, while b_1 – that of 3 rules. Thus, $\boldsymbol{b}_3$ is more compact and can be considered to fit better to the "min" criterion in (18).
That is, $\boldsymbol{b}_3\mathbf{1}^T = 2 < 3 = \boldsymbol{b}_1\mathbf{1}^T$.

8. Compacting the representation

The binary representation $\mathcal{B}(G)$ is long enough to make a practical application difficult. For example, if $\Upsilon = 26$ (the length of the English alphabet), then the number of representation bits is $\Xi = \Upsilon^3 + \Upsilon^4 = 456552$. Some techniques to reduce the domain dimension seems crucial for the problem.

An expected grammar should not contain too many production rules. Thus, the binary representation might be a sparse vector, i.e. only a very limited number of its bits might be nonzero. If so, the representation can be shrunk to a map on a limited domain of bit indices taking binary values. It is only necessary for nonzero bits to be represented in such a sparse vector. Anyway, the zeros are also allowed there. Let $\mathcal{v}$ denote a **sparse quasi-vector** (called sparse vector, in short) for $\mathcal{B}(G)$. Then let

$$\mathcal{v}: I \to \{0,1\}, \tag{21}$$

where

$$\underline{I} = \{i \mid b_i \neq 0\} \subseteq I \subseteq \{1, 2, \dots, \Xi\}, \tag{22}$$
$$\mathcal{v}(i) = b_i, \tag{23}$$

for $\mathcal{B}(G)$ of the form as per (1) with $n = \Xi$. $\underline{I}$ will be called **support**. Of course, there might exist more than one sparse quasi-vector for a particular $\mathcal{B}(G)$. That means, a sparse vector $\mathcal{v}$ can be extended to new indices that map to the value of 0, i.e. we can add a new index to I, while $\mathcal{v}$ takes 0 on it. This way, a sparse binary representation can be of use in practical applications.

For some reasons, bits can be organized into pairwise disjoined sets covering all the nonzero values. Let $\mathcal{D}_{\mathcal{v}}$ denote the domain of $\mathcal{v}$, i.e. for $\mathcal{v}$ as in (20) this is $\mathcal{D}_{\mathcal{v}} = I$. A **sparse multi quasi-vector** $\boldsymbol{\mathcal{v}}$ is a family of sparse quasi-vectors $\mathcal{v}_k$ over an index set $\mathcal{K}$,

$$\boldsymbol{\mathcal{v}} = \{\mathcal{v}_k\}_{k\in\mathcal{K}}, \tag{24}$$

where each $k \in \mathcal{K}$ can be interpreted as a "subgroup" representative, and subject to

$$\mathcal{D}_{\mathcal{v}_{k_1}} \cap \mathcal{D}_{\mathcal{v}_{k_2}} = \emptyset, \tag{25}$$

for $k_1, k_2 \in \mathcal{K}$ and $k_1 \neq k_2$, and

$$\{i \mid b_i \neq 0\} \subseteq \bigcup_{k\in\mathcal{K}} \mathcal{D}_{\mathcal{v}_k} \subseteq \{1, 2, \dots, \Xi\}. \tag{26}$$

Example 5
Let $\boldsymbol{b}_1$ be as in example 1. One of its possible compact representations is: the domain $I = \{7,20,33\} = \underline{I}$; and the sparse vector expressed by the mapping on I where $\mathcal{v}(7) = \mathcal{v}(20) = \mathcal{v}(33) = 1$. Of course, the sparse vector might be extended to $\mathcal{v}'$ on domain $I' = \{1,2,\dots 33\}$, where $\mathcal{v}'(7) = \mathcal{v}'(20) = \mathcal{v}'(33) = 1$ and $\mathcal{v}'(i) = 0$ for $i \notin \{7, 20, 33\}$.

Example 6
Let further $\boldsymbol{b}_1$ be as in example 1.
Let $I' = \{1, \dots, 10\}$, $I'' = \{11, \dots, 20\}$, $I''' = \{31, \dots, 40\}$, $\mathcal{v}': I' \to \{0,1\}$ such that $\mathcal{v}'(i) = 1$ for $i = 7$ and 0 – otherwise, $\mathcal{v}'': I'' \to \{0,1\}$ such that $\mathcal{v}''(i) = 1$ for $i = 20$ and 0 – otherwise, $\mathcal{v}''': I''' \to \{0,1\}$ such that $\mathcal{v}'''(i) = 1$ for $i = 33$ and 0 – otherwise. $\boldsymbol{\mathcal{v}} = \{\mathcal{v}', \mathcal{v}'', \mathcal{v}'''\}$ is one of the possible sparse multi vectors representing $G_1 = \mathcal{B}^{-1}(\boldsymbol{b}_1)$. The domains of the sparse vectors belonging to it can cover more than just the support $\underline{I}$.

9. Conclusion

The idea outlined here in this paper is to find a grammar for some unknown text (considered "sentence", in terms of the theory of languages). It is not necessary for the resulting grammar to be exact, though. The grammar structure can be "approximated" in the above routine, yet still giving some overview of the properties of this unknown language. The properties are meant to be somehow "hidden" in this structure, but its interpretation is beyond the scope of this article.

Further research should be undertaken to find parsing algorithms that will allow to construct appropriate fitting functions. That is, it should be easy to compute a fitting function value based on such algorithms. Moreover, it is desirable that some partial results from such algorithms should be used for computing the fitting function value for the nearest arguments, so as to speed up the whole routine. Approximate, or heuristic, algorithms might also be of use. This is due to the fact that, as already mentioned, an approximate grammar structure might be sufficient for some purposes like e.g. comparing two different styles of writing.

Having built the fitting function, optimisation methods should be chosen to solve the problems shown in this article. Binary optimisation problems have well known methods to solve. However, in view of the fact that the grammar finding problem requires a lot of binary variables, some heuristic methods, like e.g. quantum-inspired genetic algorithms, should be considered first.

10. Bibliography

[1] Boyd S., Vandenberghe L., *Convex Optimization*, 7th Edition, Cambridge University Press, 2009.

[2] Chudy M., *Wybrane algorytmy optymalizacji*, EXIT, 2014.

[3] Foryś M., Foryś W., *Teoria automatów i języków formalnych*, EXIT, 2005.

[4] Galas Z., Nykowski I., Żółkiewski Z., *Programowanie wielokryterialne*, PWE, 1987.

[5] Hopcroft J.E., Motwani R., Ullman J.D., *Introduction to Automata Theory, Languages and Computation*, 2nd Edition, Pearson Education, 2001.

[6] Kusiak J., Danielewska-Tułecka A., Oprocha P., *Optymalizacja*, PWN SA, 2009.

[7] Linz P., *An Introduction to Formal Languages and Automata*, 5th Edition, Jones & Bartlett Learning, 2012.

[8] Martin J.C., *Introduction to Languages and the Theory of Computation*, 4th Edition, McGraw-Hill, 2011.

[9] Taha H. A., *Operations research: An Introduction*, 8th Edition, Pearson Education, 2008.

[10] Woods W.A., "Context-Sensitive Parsing", *Communications of the ACM*, Vol. 13, No. 7, 437–445 (1970).

[11] Younger D.H., "Recognition and Parsing of Context-Free Grammars in Time n^3", *Information and Control*, Vol. 10, Issue 2, 189–208 (1967).

Odkrywanie gramatyki dla nieznanego tekstu jako zadanie optymalizacyjne

P.A. RYSZAWA

Niniejszy artykuł dotyczy problemu odkrywania nieznanej gramatyki z próbki tekstu. Metody odkrywania zostały sformułowane jako zadania optymalizacyjne oparte na binarnej reprezentacji gramatyk kontekstowych. Reprezentacja ta, początkowo jako najdłuższy możliwy wektor bitów, ostatecznie została skrócona do zwięzłej postaci, nadającej się do wykorzystania praktycznego. Dla uproszczenia uwzględniono tylko gramatyki nieskracające rzędu 2, z wyłączeniem produkcji postaci $P: A \to B$ i tych wyprowadzających łańcuch pusty, tj. $P: A \to \varepsilon$.

Słowa kluczowe: język kontekstowy, gramatyka formalna, gramatyka nieskracająca, reprezentacja binarna, zadanie optymalizacyjne.