Using a Genetic Algorithm to Learn a Multiple Context-Free Grammar

Artyom Skrobov

Submitted as a Seminar Paper in Learning Seminar
by Dr. Roni Katzir, Tel Aviv University, 2021

1 Introduction

The framework of *context-free grammars*, described by Chomsky (1956), is reasonably fit for formalizing English, but the CFGs are unable to cope with *cross-serial dependencies*, such as in the copy language $\{ww \mid w \in \{a,b\}^+\}$ or in the powers language $a^n b^n c^n$. Structures equivalent to the copy language, and therefore inexpressible within CFGs, had been observed in natural languages such as Dutch and Swiss German: (Shieber (1985))

...	*das*	*mer*	*d'chind*	*em Hans*	*es huus*	*lönd*	*hälfe*	*aastriiche*
	that	we	the children-ACC	Hans-DAT	the house-ACC	let	help	paint

'... that we let the children help Hans paint the house'

In 1987, Kasami *et al.* (1987) suggested a generalization of CFGs, named by them *multiple CFGs*. Similar frameworks, known as *linear context-free rewriting systems* and *multicomponent tree adjoining grammars*, are equivalent in their expressive power to MCFGs, as shown by Vijay-Shanker *et al.* (1987). Figure 1 shows the MCFG/LCFRS G_{copy} which the further discussion refers to.

Unlike in a classic CFG, the non-terminals in an MCFG represent predicates of a fixed dimension, accepting strings as arguments. CFGs, then, are a special case of MCFG in which all NTs are unary predicates. Each production rule is

$$S(XY) \leftarrow P(X,Y) \tag{1}$$
$$P(a,a) \leftarrow \varepsilon \tag{2}$$
$$P(b,b) \leftarrow \varepsilon \tag{3}$$
$$P(XY,ZW) \leftarrow P(X,Z), P(Y,W) \tag{4}$$

Figure 1: The grammar G_{copy} for the copy language $\{ww \mid w \in \{a,b\}^+\}$

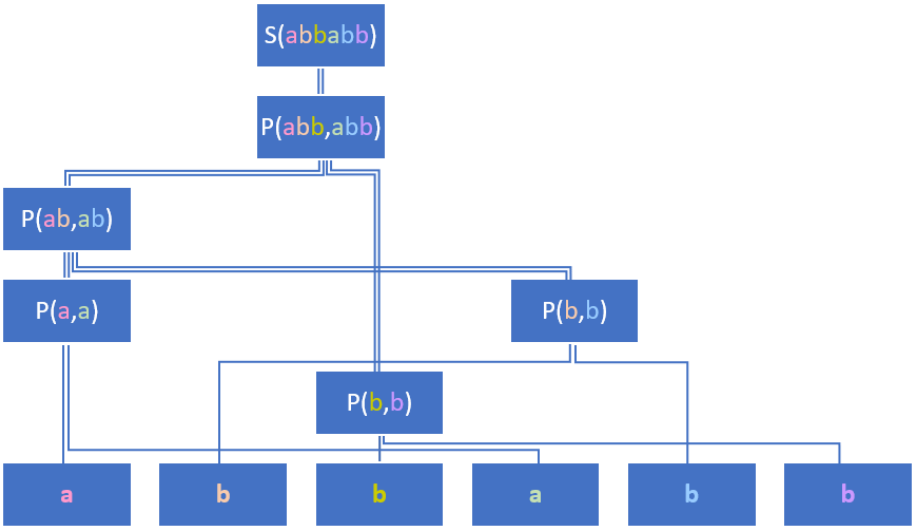

Figure 2: The parse tree for the string *abbabb* using G_{copy}

a Horn clause (Horn (1951)), asserting that if all predicates on the right-hand side are true, then so is the predicate on the left-hand side. The order of the predicates on the RHS is not significant. If the RHS is empty, then the LHS is unconditionally true; such rules are called "terminal rules". For example, the rule (2) asserts that a pair of strings (a, a) satisfies the binary predicate P; and the rule (1) asserts that if a pair of strings (X, Y) satisfies P, then their concatenation XY satisfies the unary predicate S.

We follow Clark (2014) in using the left-pointing arrow between the two sides of an MCFG production rule; others (e.g. Kallmeyer & Maier (2013)) prefer the right-pointing arrow, which may be confusing as the predicates in the RHS imply those in the LHS, and arrows denoting implication are typically directed from the premise to the conclusion.

An example of a parse tree using G_{copy} is shown in Figure 2: each edge denotes one variable in the corresponding production rule, so that unary predicates have one edge coming out of them towards S, binary predicates have two edges, etc. Each internal node corresponds to a production, i.e. inferring, according to one of the grammar's rules, that specific string(s) satisfy a specific predicate. For example, the rule (3) allows creating a $P(b, b)$ node out of any two b terminals in the input string.

Formally, an MCFG is defined as (N, T, V, S, R), where:

- N, T and V are disjoint finite sets of non-terminal, terminal, and variable symbols, correspondingly; for each $A \in N$, $dim(A)$ is a positive integer;
- $S \in N$ is the start symbol, and $\dim(S) = 1$;
- R is a finite set of rules of the form

$$A_0(Y_1, \ldots, Y_{\dim(A_0)}) \leftarrow A_1(X_1^{(1)}, \ldots, X_{\dim(A_1)}^{(1)}), \ldots, A_m(X_1^{(m)}, \ldots, X_{\dim(A_m)}^{(m)})$$

—where $A_i \in N$ for $0 \leq i \leq m$, $X_j^{(i)} \in V$ for $1 \leq i \leq m$, $1 \leq j \leq \dim(A_i)$, $Y_j \in \{T \cup V\}^+$ for $1 \leq j \leq \dim(A_0)$, and additionally, denoting Y_j as $y_j^{(1)} \ldots y_j^{(n_j)}$:

$$\{y_j^{(i)} \mid 1 \leq j \leq \dim(A_0),\ 1 \leq i \leq n_j\} \cap V = \{X_j^{(i)} \mid 1 \leq i \leq m,\ 1 \leq j \leq \dim(A_i)\}$$

$$X_{j_1}^{(i_1)} = X_{j_2}^{(i_2)} \iff i_1 = i_2 \wedge j_1 = j_2$$

$$y_{j_1}^{(i_1)} = y_{j_2}^{(i_2)} \in V \implies i_1 = i_2 \wedge j_1 = j_2$$

These additional conditions mean that each variable must be used exactly once on each side of the rule: this is the "linearity" which the name of LCFRS refers to, and it means that each substring of the input string is used exactly once in constructing the internal nodes of the parse tree. Arguably, the ellipsis in NLs can be thought of as a non-linearity in the grammar: e.g., in *Fred took a picture of you, and Susan of me*, both conjuncts include the same $[\textit{took a picture}]_{\text{VP}}$ as the predicate.

The goal of this work was to automate finding an MCFG that is the best match for the given set of strings (*corpus*). This is the same goal as that of Keinan (2020), whose work unfortunately hasn't demonstrated an ability to learn a grammar that is appropriate for the given corpus.

1.1 Related Works

Learning a classic CFG from a given corpus had been attempted many times since the 1990s, using such Machine Learning techniques as Genetic Algorithms (Lankhorst (1994)), Simulated Annealing (Katzir (2014); Keinan (2020)), Recurrent Neural Networks with external stack memory (Das *et al.* (1992)) and, more recently, without it (Cartling (2008)); and others. Yet, researchers haven't so far pursued learning an MCFG, which is a more powerful and more expressive language formalization, making the search space substantially larger. Avraham (2017) used Simulated Annealing to infer certain properties of a Minimalist Grammar, which is another formalization framework going beyond the expressive power of CFGs; but he didn't aim to learn a whole grammar.

The novelty of this work includes both technical improvements, allowing the learner to acquire a grammar more successfully than that of Keinan (2020), – and an analysis of the learner's shortcomings that make it perform much better with some corpora than with others.

2 Experimental setup

2.1 Fitness function

The first decision in constructing a learner is to define what it means for a grammar to be "the best match for the given corpus". This work uses Minimum Description Length, introduced by Rissanen (1978), as the cost function: a grammar describes the most regularities in the input data if the binary encoding of the grammar, together with the binary encoding of the input data using this grammar, takes up the fewest bits. The two components of the MDL score are conventionally designated as $|G|$ and $|D : G|$ correspondingly. All calculations allow fractional bit lengths, following the definition of entropy by Shannon (1948): e.g. when the character set comprises five characters, then each character carries $\log_2 5 \approx 2.3$ bits of information.

To calculate the MDL score, it is important exactly how the grammars and the data are encoded. For MCFGs, the following encoding is used, which adds two special characters `$` and `#` to the character set $N \cup T \cup V$: LHS uses `$` as the separator and `#` as the terminator; RHS uses `#` as the predicate terminator, and an additional `#` as the rule terminator. G_{copy}, specified in Figure 1, is encoded into the following string: (line breaks added for clarity)

```
SXY#PXY##
Pa$a##
Pb$b##
PXY$ZW#PXZ#PYW##
```

Now, there are restrictions on which characters may appear in each position: the first character in each rule is a NT, followed by a sequence of "LHS characters" out of $T \cup V \cup \{\texttt{\$}\}$, then a `#`; thereafter, each NT is followed by a sequence of variable symbols, then a `#`. Therefore, each NT takes $\log_2(|N|+1)$ to encode, including the option for the rule terminator; each LHS character takes $\log_2(|T|+|V|+2)$ to encode, including the option for the LHS terminator; and each variable takes $\log_2(|V|+1)$ to encode, including the option for the predicate terminator. The encoding of G_{copy} includes 11 NT characters, each $\log_2 3 \approx 1.6$ bits long; 17 LHS characters, each 3 bits long; and 9 variable characters, each $\log_2 5 \approx 2.3$ bits long. In total, $|G_{copy}| \approx 74.7$ bits.

Encoding a string requires, for its derivation by means of the grammar, specifying at each step which production is applied. Thus, an NT with two alternative productions requires a single bit to encode; with three possible productions, 1.6 bits; and if there's only one production possible, then no bits are necessary. For example, the string *abab* is derived as $S \to P \to PP \to aPa \to abab$ applying the sequence of rules (1), (4), (2), (3). The first production is compulsory, and takes no bits to encode; each of the others takes 1.6 bits, for a total of $|abab : G_{copy}| \approx 4.8$ bits. More generally, it can be seen that a string of length $2n$ takes $(2n-1)\log_2 3$ bits to encode using G_{copy}.

To see how MDL favors the most "meaningful" grammar, consider the corpus

$$
\begin{aligned}
P(a) &\leftarrow \varepsilon \\
P(b) &\leftarrow \varepsilon \\
S(XY) &\leftarrow P(X), S(Y) \\
S(X) &\leftarrow P(X)
\end{aligned}
$$

Figure 3: The trivial grammar G_{con} (for "concatenation")

consisting of the 40 strings $\{aa, bb, aaaa, abab, baba, bbbb, aaaaaa, aabaab, \ldots\}$, sampled from the aforementioned copy language. (The entire corpus is listed in the Appendix.) The total length of all input strings is 306 characters. The target grammar G_{copy} takes up 74.7 bits, and allows encoding the corpus in $(306 - 40) \log_2 3 \approx 421.6$ bits. On the other hand, the trivial grammar G_{con}, shown in Figure 3, takes up only $11 \log_2 3 + 9 \log_2 6 + 6 \log_2 3 \approx 50.2$ bits, but then the corpus requires 612 bits to be encoded: each terminal symbol is encoded verbatim, taking two bits, and making no use of the regularities in the input language. G_{con} allows encoding any string whatsoever, over-generalizing the input language to $\{a, b\}^+$. Finally, the "rote" grammar, including a production rule for every string in the corpus:

$$
\begin{aligned}
S(aa) &\leftarrow \varepsilon \\
S(bb) &\leftarrow \varepsilon \\
S(aaaa) &\leftarrow \varepsilon \\
S(abab) &\leftarrow \varepsilon \\
&\ldots
\end{aligned}
$$

—allows encoding the corpus in just 212.9 bits (at $\log_2 40 \approx 5.3$ bits per string), but then the grammar's own encoding takes up $40 \cdot 2 + (306 + 40) \cdot 2 = 772$ bits. G_{rote} doesn't allow encoding any strings which aren't present in the input corpus – in other words, it doesn't generalize the corpus to a bigger language.

To summarize:

grammar	$\lvert G \rvert$	$\lvert D : G \rvert$	MDL score
G_{copy}	74.7	421.6	496.3
G_{con}	50.2	612.0	662.2
G_{rote}	772.0	212.9	984.9

G_{copy} clearly wins over either encoding the input data verbatim, or incorporating the whole corpus into the grammar.

To evaluate the MDL score for a grammar, it's necessary to parse every input string using the grammar. This is done using a bottom-up Cocke–Younger–Kasami parser, adapted from Kallmeyer & Maier (2013): at each step, it tries to form new non-terminal nodes by combining nodes produced at the previous step with those produced at all previous steps. The parsing terminates

when a node is produced for the start symbol S and the whole input string. This exhaustive-search strategy incurs a very high computational complexity, amounting to $O(n^{d\cdot(r+1)})$, where d is the dimension of the MCFG, i.e. the maximum dimension among all its NTs; and r is the rank of the MCFG, i.e. the longest concatenation on the LHS among all production rules. In order to keep the parsing reasonably fast, the GA is disallowed to spawn MCFGs with $r > 4$.

More specifically, a parser for *probabilistic* MCFG is used, which, given the probability associated with each production rule, finds the probability P of the likeliest derivation for the input string: then encoding the input string using the given grammar requires $(-\log_2 P)$ bits. To be evaluated, an MCFG is first converted to a PMCFG for the parser by assigning, for each NT, equal probabilities to all rules having the NT on the LHS.

2.2 Search strategy

The second important aspect of learning the best grammar is deciding on the strategy for finding it. The search algorithm chosen for this work (Genetic Algorithm with multiple "islands") may be summarized as follows:

- Each island is initialized with the trivial grammar G_{con}, shown in Fig. 3.
- On each iteration, either one or two "parents" are chosen at random (non-uniformly), and a new grammar is spawned by combining the parents' rules, copying rules from the first parent up to a random point, then from the second parent ("crossover"), and/or by applying random small changes to the grammar ("mutations").
- If the population of an island exceeds the limit, some of its grammars are killed at random (non-uniformly).
- Occasionally, each island sends a clone of one of its grammars to the next island, in a round-robin fashion, so that the local developments on each island are eventually shared with the rest of the population.

The probabilities of procreation, death, and migration are all determined by the grammar's relative rank in the island's population, rated by the MDL score.

2.3 Implementation

The learner is implemented in Python, and split into four modules:

- `Grammar.py` implements the object model for PMCFG (classes `Pred, PRule, Grammar`), the latter including the ruleset for grammar mutations;
- `Parser.py` implements a CYK parser for PMCFG, adapted from Keinan (2020): some technical improvements to his code, e.g. using Python frozen data classes, allowed to speed up parsing by an order of magnitude, while retaining the same essential algorithm;

- `GeneticAlgorithm.py` implements the GA for finding the best MCFG, including MDL scoring and a simple visualization of the evolution progress, relying on the `graphics` library for Python by Zelle (2004);
- `Main.py` sets the input corpus, initial grammar, and other meta-parameters for the GA.

A technical limitation of the PMCFG parser being used is that terminal symbols can only appear with unary NTs, on the LHS of trivial terminal rules of the form $N(t) \leftarrow \varepsilon$. This restriction, reminiscent of the Chomsky normal form for CFGs (Chomsky (1959)), doesn't affect the expressive power of MCFGs, as any MCFG can be normalized into such a form; moreover, for grammars used in NLP, such form is most natural, as each terminal symbol (corresponding to a lexicon entry) is first categorized as a part of speech, before being used in further productions.

On the other hand, the parser being used is less restrictive than the formal definition of (P)MCFG as given in the Introduction, in that it doesn't require every variable predicated on the RHS of a production rule to be used on its LHS. E.g., the rule $S(X) \leftarrow P(X)R(Y)$ means "S can be produced from P if R can be produced from any substring of the input string", and $S(X) \leftarrow P(X,Y)$ means "S can be produced from a substring if P can be produced from it and some other substring of the input string" – similar to the lookahead/lookbehind features of extended regular expressions. It was shown by Kallmeyer & Maier (2013) that the requirement for all predicated variables to be used on the LHS doesn't affect the expressive power of MCFG; for the purposes of GA, it is helpful that mutation chains are allowed to include intermediate grammars whose rules include unused variables.

An important part of designing the GA was to define the ruleset for mutations, making sure that all mutations result in valid MCFGs, and every valid MCFG can be reached via a chain of mutations. The following 15 rules are used:

1. Add a new terminating rule for an existing or a new NT;
2. Add an aliasing rule between two NTs of the same dimension;
3. Add a concatenating rule with two unary NTs on the RHS;
4. Delete a random rule;
5. Expand a unary NT using a random production rule;
6. Add a random NT on the RHS, ignore its predicated variables on the LHS;
7. Delete a NT whose predicated variables are ignored on the LHS;
8. Delete the use of a random variable on the LHS;
9. Concatenate an unused variable in a random position on the LHS;
10. Split a NT, adding a new dimension, e.g. $P(XY) \leftarrow A(X)B(Y),\ S(X) \leftarrow P(X)$ into $P(X,Y) \leftarrow A(X)B(Y),\ S(XY) \leftarrow P(X,Y)$;
11. Merge two dimensions of a NT (i.e. the inverse of the above);
12. Swap two dimensions of a multidimensional NT on the LHS;
13. Swap two variables in a concatenation on the LHS;
14. Mutate the terminal symbol in a terminal rule;
15. Mutate a NT symbol on either side of a rule.

3 Experimental results

The settings used for the GA in the experiments were as follows: 10,000 iterations; 10 islands of up to 200 grammars each; mortality rate 50% for the lowest scored grammar, decreasing exponentially with the rank; mutation rate 60%; crossover rate 20%; and migrations every 100 iterations. If a grammar fails to produce some of the words in the training set, its $|D : G|$ score receives a penalty of 35 bits for each such word. Before calculating $|G|$, each grammar is stripped of all non-reachable rules and NTs, and the penalty for having them is $(|G_{original}| - |G_{stripped}|)/100$ – in most observed cases, the penalty amounts to a few tenths of a bit. This allows "dormant" rules, facilitating multi-stage improvement, to remain in the best-performing grammars, while still applying a slight pressure to get rid of them.

Figure 4 illustrates the evolution of the grammars: each of the 10 graphs corresponds to an island, the green line is the best score in the population, the red is the worst, and the brown is the average. Each horizontal pixel is one generation, for a total of 1800×10 generations, and each vertical pixel is 50 score points. During the first 200 generations, no grammars die, and the average score gradually worsens. Then, over a few hundred generations, bad grammars die out, and the average score gradually improves. After such a "ramp-up", the average score approaches the best, and the worst shows increasingly erratic fluctuations corresponding to newly spawned bad grammars staying alive for up to a dozen or two generations.

The GA took 4,088 iterations to arrive at a grammar equivalent to the following CFG, which will be referred to as G_{opt}:

$$A \to a \,|\, b$$
$$S \to AA \,|\, ASA$$

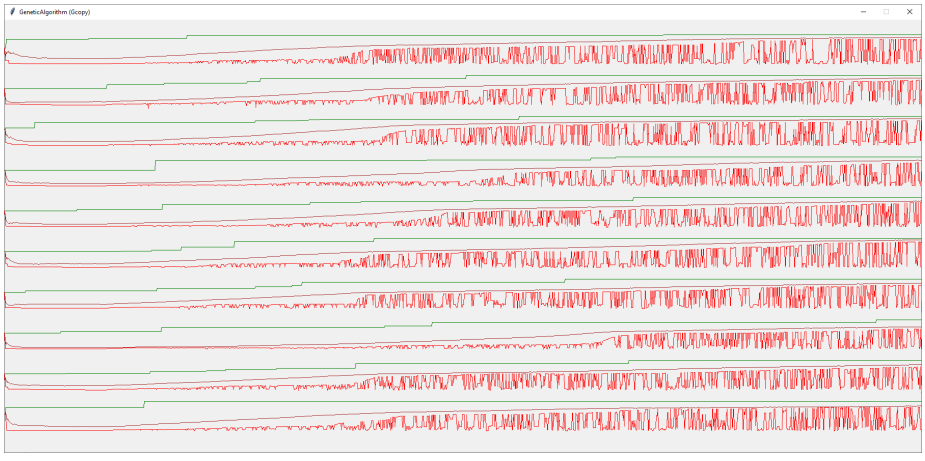

Figure 4: Evolution targeting G_{copy}

G_{opt} expresses only the generalization that all input strings have even length. Unexpectedly, $|G_{opt}| = 71.1$ and $|D : G_{opt}| = 459$, so that the resulting MDL score of 530.1 bits beats the target G_{copy} by a small margin. The reason is that its savings on being very simple outweigh its losses on over-generalizing the input language. In order to account for this, the score calculation has been changed from pure MDL to $|G| + 3 \times |D : G|$, as if the corpus had three copies of each input string, so that the over-generalization incurs three times the loss. This way, the target grammar's score remains lower than that of the distractor:

grammar	$\|G\|$	$\|D : G\|$	$\|G\| + \|D : G\|$	$\|G\| + 3 \times \|D : G\|$
G_{con} (initial)	48.6	612.0	660.6	1884.6
G_{opt} (result)	71.1	459.0	530.1	1448.1
G_{copy} (target)	144.3	421.6	565.9	1409.1

Even with the updated scoring, the best found grammar $G_{opt'}$ remained weakly-equivalent to G_{opt}, describing the same effective language $(a|b)^{2n}$ – but $G_{opt'}$ allows shallower derivations than G_{opt}, thus lowering the description length for longer input strings at the expense of shorter ones:

$$
\begin{aligned}
A &\to PP \\
P &\to a \,|\, b \\
C &\to A \,|\, AA \\
S &\to C \,|\, ASA
\end{aligned}
$$

$|G_{opt'}| = 135.7$ and $|D : G_{opt'}| = 434$, therefore its score 1437.7 is still above the target. Yet, the target grammar – in fact, any MCFG making use of multidimensional NTs and thereby transcending the expressive power of classic CFGs – wasn't reached during the experiment, which took a couple of days to run on my laptop.

To assess whether the failure is due to peculiarities of the specific target language, or the search space of MCFGs in general, or the implementation and settings of the GA, – experiments were run, using the same settings, with three other corpora, sampled from the language of palindromes $\{a, b, aa, bb, aaa, aba, bab, \ldots\}$ (27 strings in total), and from two powers languages, $\{ab, aabb, aaabbb, aaaabbbb, aaaaabbbbb\}$ and $\{abc, aabbcc, aaabbbccc, aaaabbbbcccc\}$.

4 Discussion

To see why G_{copy} hadn't been reached, I've manually constructed two mutation chains, shown in Figure 5: each column shows the two components of the MDL score for an intermediate grammar along the chain. (To make variations in both $|G|$ and $|D : G|$ discernible, the scores are plotted on a logarithmic scale.) These two chains are certainly not the only ones possible, and they may not be the shortest either; but they do illustrate the difference in "steepness" between the two transitions: in the chain $G_{con} \to G_{opt}$, the longest "uphill walk" is two

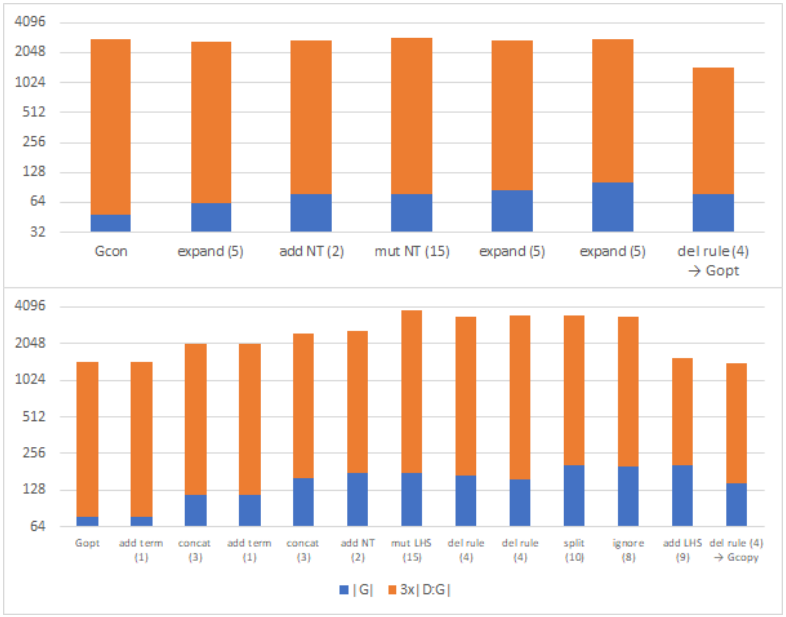

Figure 5: Mutation chains $G_{con} \to G_{opt}$ and $G_{opt} \to G_{copy}$

steps, incurring a 12% increase in score; whereas in $G_{opt} \to G_{copy}$, it is six steps, incurring a $\times 2.7$ increase. It seems natural that the latter chain is too steep for the GA to climb.

The results for the three other corpora have been somewhat more successful: it is noteworthy that the very small samples from the powers languages didn't lead to emergence of "rote" grammars.

- For the language $a^n b^n$, the learner easily found the target grammar $G_{a^n b^n}$, equivalent to the CFG $A \to a,\ B \to b,\ S \to AB \,|\, ASB$:

grammar	$\lvert G\rvert$	$\lvert D:G\rvert$	$\lvert G\rvert + \lvert D:G\rvert$	$\lvert G\rvert + 3 \times \lvert D:G\rvert$
G_{con} (initial)	48.6	60.0	108.6	228.6
$G_{a^n b^n}$ (target)	74.9	15.0	89.9	119.9

- For the language $a^n b^n c^n$, the best grammar G_{result} found by the learner was equivalent to the following CFG:

$$
\begin{aligned}
A &\to a \\
B &\to b \\
C &\to c \\
P &\to PB \,|\, C \\
S &\to ASP \,|\, b
\end{aligned}
$$

—whose effective language is $a^n b(b^* c)^n$. The initial grammar $G_{con'}$ was different from the other experiments in that it allowed three terminal characters $\{a, b, c\}$. The target MCFG $G_{a^n b^n c^n}$:

$$\begin{aligned} A(a) &\leftarrow \varepsilon \\ B(b) &\leftarrow \varepsilon \\ C(c) &\leftarrow \varepsilon \\ P(X, Y, Z) &\leftarrow A(X), B(Y), C(Z) \\ P(XU, YV, ZW) &\leftarrow A(X), B(Y), C(Z), P(U, V, W) \\ S(XYZ) &\leftarrow P(X, Y, Z) \end{aligned}$$

—was not found. This time, the target grammar is so complex that even the $3\times$ factor wasn't enough to bring its MDL score below that of G_{result}:

grammar	$\lvert G\rvert$	$\lvert D : G\rvert$	$\lvert G\rvert + \lvert D : G\rvert$	$\lvert G\rvert + 3 \times \lvert D : G\rvert$
$G_{con'}$ (initial)	59.4	112.0	171.4	395.4
G_{result}	100.5	30.0	130.5	190.5
$G_{a^n b^n c^n}$ (target)	192.9	10.0	202.9	222.9

- Last, for the language of palindromes, the target grammar equivalent to the CFG

$$\begin{aligned} A &\to a \\ B &\to b \\ S &\to ASA \,|\, BSB \,|\, AA \,|\, BB \,|\, a \,|\, b \end{aligned}$$

—was not found; instead, the learner optimized the initial grammar G_{con} into $G_{opt''}$ which is equivalent to the CFG

$$\begin{aligned} P &\to a \,|\, b \\ C &\to P \,|\, PP \\ S &\to C \,|\, PSP \end{aligned}$$

—and allows shallower derivations than G_{con} while describing the same effective language $\{a, b\}^+$. This case, same as for the copy language, may be considered the learner's failure, as the target grammar has lower MDL score than that of the found grammar:

grammar	$\lvert G\rvert$	$\lvert D : G\rvert$	$\lvert G\rvert + \lvert D : G\rvert$	$\lvert G\rvert + 3 \times \lvert D : G\rvert$
G_{con} (initial)	48.6	264.0	312.6	840.6
$G_{opt''}$ (result)	106.1	233.0	339.1	805.1
G_{pal} (target)	151.8	191.3	343.0	725.6

To conclude, we may estimate that MCFGs present difficulties for GA because of the non-local constraints on a grammar's validity: e.g. all appearances of a NT must have the same dimension, the two sides of a production rule must

have the same variables without omissions or repetitions, etc. As a result, the mutation rules have to be quite complex: e.g. the split rule (#10) involves updating all productions where the NT appears on either side. On the other hand, classic CFGs, where any string of grammar symbols is a valid RHS for any production, are trivial to mutate; and so, small mutations tend to cause small changes to the grammar's performance. As the GA in the experiments has never found multidimensional NTs useful, effectively it was searching in the subspace of classic CFGs, but using the MCFG mutation rules.

5 Further work

Possible improvement of the learner may follow these two directions: the first is to use, instead of MCFG, an equally expressive formalism whose representation of a grammar may have less internal structure, making it more amenable to mutations. Minimalist Grammars, used by Avraham (2017), may (or may not) be such an option. The other direction, *a priori* seeming more promising, is to find a robust way of choosing the meta-parameters for the GA, such as population sizes, mutation rates, longevity of individual grammars, etc. For the experiments described in this work, all meta-parameters values were picked based on intuition alone, and their effect had not been rigorously evaluated; the only observation has been that slight changes don't have any noticeable effect.

Appendix: Code and input data

The code of the learner is available on GitHub: `https://github.com/tyomitch/pmcfg`; for visualization, it uses the `graphics` library by Zelle (2004), available at `https://mcsp.wartburg.edu/zelle/python/graphics.py`

The following 40 strings comprise the corpus for the copy language: $\{aa, bb, aaaa, abab, baba, bbbb, aaaaaa, aabaab, abaaba, baabaa, abbabb, bbabba, bbbbbb, aaaaaaaa, aaabaaab, aabaaaba, abaaabaa, baaabaaa, aabbaabb, abbaabba, bbaabbaa, abbbabbb, bbbabbba, abababab, babababa, bbbbbbbb, aaaaaaaaaa, aaaabaaaab, aaabaaaaba, aabaaaabaa, abaaaabaaa, baaaabaaaa, aaabbaaabb, aabbaaabba, abbaaabbaa, bbaaabbaaa, aabbbaabbb, abbbaabbba, bbbaabbbaa, abbbbabbbb\}$

The following 27 strings comprise the corpus for the language of palindromes: $\{a, b, aa, bb, aaa, aba, bab, bbb, aaaa, abba, baab, bbbb, aabaa, baaab, abbba, abaaba, abbabba, abbaabba, abababa, aabaaabaa, babab, babbab, abaaaba, aaabaaa, aabbbaa, abbbba, babbbbab\}$

The corpora used for the two powers languages are $\{ab, aabb, aaabbb, aaaabbbb, aaaaabbbbb\}$ and $\{abc, aabbcc, aaabbbccc, aaaabbbbcccc\}$.

References

Avraham, Tomer. 2017. *Learning Head-Complement Order with Minimalist Grammars*. Ph.D. thesis, Tel Aviv University.

Cartling, Bo. 2008. On the implicit acquisition of a context-free grammar by a simple recurrent neural network. *Neurocomputing*, **71**(7-9), 1527–1537.

Chomsky, Noam. 1956. Three models for the description of language. *IRE Transactions on information theory*, **2**(3), 113–124.

Chomsky, Noam. 1959. On certain formal properties of grammars. *Information and control*, **2**(2), 137–167.

Clark, Alexander. 2014. *An introduction to multiple context free grammars for linguists*.

Das, Sreerupa, Giles, C Lee, & Sun, Guo-Zheng. 1992. Learning context-free grammars: Capabilities and limitations of a recurrent neural network with an external stack memory. *In: Proceedings of The Fourteenth Annual Conference of Cognitive Science Society. Indiana University*, vol. 14. Citeseer.

Horn, Alfred. 1951. On sentences which are true of direct unions of algebras1. *The Journal of Symbolic Logic*, **16**(1), 14–21.

Kallmeyer, Laura, & Maier, Wolfgang. 2013. Data-driven parsing using probabilistic linear context-free rewriting systems. *Computational Linguistics*, **39**(1), 87–119.

Kasami, Tadao, Seki, Hiroyuki, & Fujii, Mamoru. 1987. Generalized context-free grammars, multiple context-free grammars and head grammars. *Preprint of WG on Natural Language of IPSJ*.

Katzir, Roni. 2014. A cognitively plausible model for grammar induction. *Journal of Language Modelling*, **2**.

Keinan, Dan. 2020. *A PMCFG MDL Syntactic Learner*.

Lankhorst, Marc M. 1994. A genetic algorithm for the induction of context-free grammars. *Pages 87–100 of: CLIN IV. Papers from the Fourth CLIN Meeting*.

Rissanen, Jorma. 1978. Modeling by shortest data description. *Automatica*, **14**(5), 465–471.

Shannon, Claude Elwood. 1948. A mathematical theory of communication. *The Bell system technical journal*, **27**(3), 379–423.

Shieber, Stuart M. 1985. Evidence against the context-freeness of natural language. *Pages 79–89 of: Philosophy, language, and artificial intelligence*. Springer.

Vijay-Shanker, Krishnamurti, Weir, David, & Joshi, Aravind K. 1987. Characterizing structural descriptions produced by various grammatical formalisms. *In: Proceedings of the 25th Annual Meeting of the Association For Computational Linguistics (ACL).*

Zelle, John M. 2004. *Python programming: an introduction to computer science.* Franklin, Beedle & Associates, Inc.