ISSN 0103-9741

Monografias em Ciência da Computação

nº 04/2021

An extension to the minimal grammar problem

Alex Garcia

Edward Hermann Haeusler

Departamento de Informática

Monografias em Ciência da Computação, No. 04/2021 ISSN: 0103-9741
Editor: Prof. Carlos José Pereira de Lucena Setembro, 2021

An extension to the minimal grammar problem

Alex Garcia and Edward Hermann Haeusler

garcia@ime.eb.br, hermann@inf.puc-rio.br

Abstract. The smallest grammar problem consists of finding the smallest CFG that generates only a given string. This is an NP-hard problem, and there are various heuristics to solve it described in the literature. In this paper, we propose an extension to this problem, allowing the grammars to have a reverse operator and a complement operator. We argue that the heuristics used to find approximate solutions to the smallest grammar problem can be easily adapted to these extensions. Experiments confirm that these extensions perform better than plain CFGs when applied to virus genome compression.

Keywords: Smallest Grammar Problem; Data Compression; Hierarchical Structure Inference; approximation algorithm, Sequitur, IRR.

Resumo. O problema da menor gramática consiste em determinar a menor gramática livre de contexto que gera um único string dado como entrada. Este problema é NP-difícil, existem várias heurísticas para resolvê-lo descritas na literatura. Neste trabalho propomos estender este problema, permitindo que as gramáticas tenham um operador de reverso e um operador de complemento. Argumentamos que as heurísticas usadas para encontrar soluções aproximadas para o problema da menor gramática podem ser facilmente adaptadas para estas extensões. Experimentos confirmam que estas extensões apresentam melhores resultados do que as GLCs tradicionais quando aplicadas à compressão de genoma de vírus.

Palavras-chave: Problema da menor gramática; Compressão de dados; Inferência de estrutura; algoritmo de aproximação, Sequitur, IRR.

In charge of publications:
PUC-Rio Departamento de Informática - Publicações
Rua Marquês de São Vicente, 225 - Gávea
22453-900 Rio de Janeiro RJ Brasil
Tel. +55 21 3527-1516 Fax: +55 21 3527-1530
E-mail: publicar@inf.puc-rio.br
Web site: http://bib-di.inf.puc-rio.br/techreports/

Table of Contents

1 Introduction

The smallest grammar problem involves finding one of the smallest context-free grammar (CFG) that generates the language with only a single given string ω. The decision version of this problem is NP-complete. This problem has been extensively researched. [7] presents a survey of various heuristics and proposes new ones.

Sequitur [18] is an efficient algorithm based on a eager heuristic to solve this problem, its authors claim that finding a grammar that generates a string entails identifying structure in the string. The authors discuss applications of this structure inference in music, natural language texts, descriptions of plants, graphical figures, DNA sequences, a genealogical database, programming languages and execution traces. Other authors mention applications such as: data compression [6] [14] [13] [20] [15] [1] [2], identify language properties [8], nucleic acid compression [9], biologically meaningful microRNA sequence identification [9], program optimization [19] and text-to-speech conversion [19].

A further advantage of using grammars as a data compression approach is that various problems regarding the generated string can be solved efficiently without decompression, [16] surveys several of them, such as:

- Equality Checking. Is the string generated by a grammar G_1 equal to the one generated by G_2? It is noteworthy that if one does not restrict the check to grammars generating single strings, then the problem is undecidable.

- Compressed Pattern Matching. Is the string generated by a grammar G_1 a substring of the one generated by G_2?

- Compressed Membership Problem. Is the string generated by a grammar G_1 in a given CFL L?

In the present work, we investigate an extension of the smallest grammar problem. We introduce augmented context-free grammar (or augmented grammar) as context-free grammar enriched with reverse and complement operators. We consider the problem of finding the smallest augmented grammar that generates a given string. It is well-known that searching for the reverse of a substring in a string has the same complexity as the search of the substring. The same is true for the complement operator, defined in section 9. Therefore, we can apply most, if not all, heuristics for solving the smallest grammar problem with little change to the augmented grammar problem with no impairment to its asymptotic complexity measures. We argue that this extension has most of the applications as the original smallest grammar problem. It sure can be applied to data compression. For this purpose, its enhanced expression capability allows the minimal augmented grammar to beat the minimal context-free grammar by a linear factor. Regarding natural language structure identification, it may be able to benefit from the inversion [4], whereas, in genetics, inversion is a type of chromosomal rearrangement [21].

2 Small Grammars

In this section, we introduce the concept of Small Grammars. This concept is very similar to *Admissible Grammars* defined in [14], the major difference is that Small Grammars may generate the empty string.

We assume the reader is familiar with a computer science presentation of CFG such as [12]. We define a Small Context-Free Grammar (SCFG) to be a CFG $G_S = (N, \Sigma, P, S)$, in which:

1. $L(G)$ has exactly one string, ω;
2. G is non ambiguous;
3. all nonterminals in N are used in any derivation $S \Rightarrow^* \omega$;

It is easy to verify that the following properties hold for any SCFG $G_S = (N, \Sigma, P, S)$.

Theorem 2.1. *Given an SCFG $G_S = (N, \Sigma, P, S)$, starting from any nonterminal $A \in N$ it is possible to generate exactly one string.*

Proof. From the SCFG definition, we know that there is exactly one string ω such that $S \Rightarrow^* \omega$, and that all nonterminals in N are used in this derivation. So this derivation can be rewritten as: $S \Rightarrow^* \alpha A \beta \Rightarrow^* \alpha\gamma\beta = \omega$. Therefore A generates at least the string γ. Suppose that also generates a different string γ_2, then $S \Rightarrow^* \alpha A\beta \Rightarrow^* \alpha\gamma_2\beta \neq \omega$, which contradicts condition 1 of the SCFG definition. Hence A generates only γ. $\square$

Definition 2.1 (Yield)**.** Given an SCFG $G_S = (N, \Sigma, P, S)$ and $v \in (\Sigma \cup N)^*$, *yield*(v) is the string defined as:

- If $a \in \Sigma$, $yield(a) = a$.
- If $A \in N$, $yield(a) = w$, where w is the single string generated by w.
- $yield(\epsilon) = \epsilon$.
- If $u \in (\Sigma \cup N)$ and $v \in (\Sigma \cup N)^*$, $yield(uv) = yield(u)yield(v)$.

Theorem 2.2. *Given an SCFG $G_S = (N, \Sigma, P, S)$, for each $A \in N$ there is exactly one rule $A \to \alpha \in P$.*

Proof. Suppose there are two different rules $A \to \alpha, A \to \beta \in P$, then if $yield(\alpha) \neq yield(\beta)$, A generates two strings, which contradicts theorem 4.1, whereas if $yield(\alpha) = yield(\beta)$, the grammar is ambiguous, therefore not an SCFG. $\square$

As a consequence of this fact we sometimes use the words *rule* and *nonterminal* indistinguishably for SCFGS.

2.1 Nonterminal order

Definition 2.2 (ρ)**.** Given a CFG $G = (N, \Sigma, P, S)$. If $A, B \in N$, and we define $B\rho_G A$ if and only of $A \Rightarrow_G^* \alpha B \beta$, where α, $\beta \in (\Sigma \cup N)^*$

Theorem 2.3. *If G is an SCFG then ρ_G is a partial order.*

Proof. Observe that because $\Rightarrow^*_G$ is reflexive and transitive, the relation ρ_G is always a reflexive and transitive relation. To prove that it is antisymmetric, suppose that Aρ_GB and Bρ_GA, for some $A, B \in N, A \neq B$. Then $A \Rightarrow^*_G \alpha_1 B \beta_1$ and $B \Rightarrow^*_G \alpha_2 A \beta_2$ then the unique derivation $S \Rightarrow^* \alpha A \beta \Rightarrow^* \alpha\gamma\beta = \omega$ can be rewritten as $S \Rightarrow^* \alpha A \beta \Rightarrow^* S \Rightarrow^* \alpha\alpha_1 B \beta_1 \beta \Rightarrow^* \alpha\alpha_1\alpha_2 A \beta_2\beta_1\beta \Rightarrow^* \alpha\alpha_1\alpha_2\gamma\beta_2\beta_1\beta$, which either generates a second string or else (if $\alpha_i = \beta_i = \epsilon$) makes the grammar ambiguous. In either case the grammar is not an SCFG. □

Because ρ_G is a partial order, we will write Aρ_GB as $A \leq B$ when G is an SCFG.

2.2 Minimal Grammars

Another simple fact, which we will care to prove is that the minimal CFG (i.e., the CFG with the smallest size) that generates a single string ω is an SCFG. But first, we have to define the size of a CFG. We will defer the analysis of the size concept to the next section. To define minimal grammars we will simply state the simplest CFG grammar size definition, used in [7]:

Definition 2.3 ($|G|$)**.** Given a CFG $G = (N, \Sigma, P, S)$, we define $|G| = \sum_{A \to \alpha \in P}(|\alpha|)$

Several algorithms in the literature use the decrease of $|G|$ as a termination condition, hence the use of different grammar size measures will make these algorithms behave differently. Unless otherwise stated, from now on we will use the measure defined above.

Theorem 2.4. *The minimal CFG that generates a single string ω is an SCFG.*

Proof. Let G_m be the minimal grammar that generates only ω. Suppose that G_m is not an SCFG then either:

1. $L(G_m) \neq \{\omega\}$. In this case G_m is not a solution to the problem.
2. G is ambiguous. In this case ω has multiple derivation trees relative to G_m, if you throw out the rules and nonterminals not used in the first tree, you have a new Grammar G_2, $G_2 < G_m$, $L(G_2) = \{\omega\}$, hence G_m is not minimal.
3. one nonterminal $A \in N$ is not used in a derivation $S \Rightarrow^* \omega$; in this case if you throw out this terminal and its associated rule, you also have a new Grammar G_2, $G_2 < G_m$, $L(G_2) = \{\omega\}$, hence G_m is not minimal.

□

3 Grammars size

In order to evaluate compression we also use the relative grammar size, $RGS(G)$ defined as the grammar size over the size of the original string, therefore lower values of $RGS(G)$ are associated to better compression.

Definition 3.1 ($RGS(G)$)**.** Given $G = (N, \Sigma, P, S)$, we define $RGS(G) = |G|/|yield(S)|$.

In the previous section we used a simple notion of grammar size, following [7]. A similar notion is give in [19] (followed by [6]).

Definition 3.2 ($|G|_r$)**.** Given a CFG $G = (N, \Sigma, P, S)$, we define $|G|_r = \sum_{A \to \alpha \in P}(|\alpha|+1)$

If $r = |P|$ then it is clear that the two definitions are related by the formula $|G|_r = |G| + r$. Note that for small context-free grammars one can separate different rules by a line break (since each nonterminal has exactly one rule), which justifies this definition.

Let us analyze the representation of the right side of a rule, in the context of the minimal grammar problem. We are given a fixed alphabet (for example, ASCII symbols) and we are supposed to find out the minimal CFG that generates an arbitrarily long string. In this context, each terminal symbol needs 1 unit (1 byte, for example) for its representation, but the representation of each nonterminal symbol would depend on the number of nonterminals. If we represent each nonterminal as N_i, i.e., an escape symbol, followed by a number in base 10 we can represent the i^{th} nonterminal with $1+\lceil log_{10}(i+1)\rceil$, then perhaps a more precise notion of size would be the following one.

Definition 3.3 ($|G|_{rep}$)**.** Given a CFG $G = (N, \Sigma, P, S)$, we define $|G|_{rep} = \sum_{A \to u_1 u_2 \ldots u_n \in P}(\sum_{i=1}^{n}(size(u_i)))$ where:

$$size(u_i) = \begin{cases} 1 & \text{if } u_i \text{ is a terminal} \\ 1 + \lceil log_{10}(j+1) \rceil & \text{if } u_i \text{ is the } j^{th} \text{ nonterminal} \end{cases} \tag{1}$$

If $n = |N|$ then it is clear that the definitions are related by the formula $|G|_{rep} < |G|(1 + \lceil log_{10}(n+1)\rceil)$. An issue of the $|G|_{rep}$ is that it depends on the particular order chosen to represent the nonterminals. As mentioned before, unless otherwise stated, we will use the simplest measure $|G|$ throughout this work.

4 Grammars with reverse operator

Definition 4.1 (Reverse)**.** For terminal strings in Σ^*, we define the reverse operator in the following way:

- $\epsilon^r = \epsilon$
- $(aw)^r = w^r a$, if $w \in \Sigma^*$ and $a \in \Sigma$

By this definition, if $\Sigma = \{a, b, c\}$ then:
$(abc)^r = c(ab)^r = cb(a)^r = cb(a\epsilon)^r = cba(\epsilon)^r = cba\epsilon = cba$

Definition 4.2 (N^R)**.** Given a set of nonterminal symbols, N, we define a set of new nonterminals $N^R = \{A^R | A \in N\}$. In this case A^R is not a reverse operator, but simply the name of the new nonterminal symbol in N^R.

Definition 4.3 (Reverse)**.** Given a string (sentential form) over the alphabet $\Gamma^R = (\Sigma \cup N \cup N^R)$, we extend the definition of the reverse operator in the following way:

- $\epsilon^r = \epsilon$
- $(aw)^r = w^r a$, if $w \in (\Gamma^R)^*$ and $a \in \Sigma$
- $(Aw)^r = w^r A^R$, if $w \in (\Gamma^R)^*$ and $A \in N$
- $(A^R w)^r = w^r A$, if $w \in (\Gamma^R)^*$ and $A^R \in N^R$

Definition 4.4 (RCFG)**.** A CFG with reverse operator (RCFG) is a tuple $G = (N, \Sigma, P, S)$, where:

- N is a finite set nonterminals;
- Σ is the alphabet of terminal symbols;
- P is a set of productions rules of the form $A \to \alpha$, where:
  - $A \in N$ is a nonterminal;
  - $\alpha \in (\Sigma \cup N \cup N^R)^*$ is a sentential form;
- $S \in N$ is the initial symbol.

Definition 4.5 (One-step derivation relation on $(\Gamma^R)^*$)**.** Given a RCFG $G = (N, \Sigma, P, S)$, we say that $w_1 \Rightarrow_G w_2$ when:

- $w_1 = \gamma_1 A \gamma_2$, $w_2 = \gamma_1 \alpha \gamma_2$, $A \to \alpha \in P$, $\gamma_1, \gamma_2, \alpha \in (\Gamma^R)^*$
- $w_1 = \gamma_1 A^R \gamma_2$, $w_2 = \gamma_1 \alpha^r \gamma_2$, $A \to \alpha \in P$, $\gamma_1, \gamma_2, \alpha \in (\Gamma^R)^*$

Definition 4.6 (Multiple steps derivation relation on $(\Gamma^R)^*$)**.** $\Rightarrow_G^*$ is the reflexive transitive closure of $\Rightarrow_G$.

When the grammar is clear from the context we write simply $\Rightarrow$ for the one-step derivation relation and $\Rightarrow^*$ for the multiple steps derivation relation.

Definition 4.7 ($L(G)$)**.** Language generated by a RCFG $G = (N, \Sigma, P, S)$:
$L(G) = \{\omega \in \Sigma^* \mid S \Rightarrow^* \omega\}$

Theorem 4.1. *Given a RCFG $G = (N, \Sigma, P, S)$, $A \Rightarrow_G^* \omega$ if and only if $A^R \Rightarrow_G^* \omega^r$.*

Proof. The theorem follows trivially from the definition of $\Rightarrow_G$. □

Since the presence of the reverse operator can be interpreted as implicit new rules, the reverse operator does not increase the power of CFG grammars, i.e., the class of languages generated by CFG and CFGR grammars is the same, namely, the context-free languages. The presence of reverse operators may though increase the compression power of a grammar.

5 Grammars with complement operator

Definition 5.1 (Complement Relation)**.** A relation on an alphabet Σ is any relation ρ on Σ such $\rho \circ \rho$ is the identity relation.

Though the definition allows many different relations in the experiments we will work only with two irreflexive relations on even size alphabets: For experiments on genetic sequences we use a four-letter alphabet with the Watson–Crick complement relation, whereas for the remaining files we use a 256 letter binary alphabet, where each byte (8-bit value) is one letter, and ρ is the relation that complements the least significant bit.

Definition 5.2 (Complement operator). For terminal strings in Σ^* and a complement relation ρ on Σ, we define the complement operator in the following way:

- $\epsilon^c = \epsilon$
- $(aw)^c = \rho(a)w^c$, if $w \in \Sigma^*$ and $a \in \Sigma$

By this definition, if $\Sigma = \{a, t, c, g\}$ and $\rho = \{(a, t), (t, a), (c, g), (g, c)\}$ then: $(gataca)^c = \rho(g)(ataca)^c = c\rho(a)(taca)^c = ct\rho(t)(aca)^c = cta\rho(a)(ca)^c = ctat\rho(c)(a)^c = ctatg\rho(a) = ctatgt$

Definition 5.3 (N^C). Given a set of nonterminal symbols, N, we define a set of new nonterminals $N^C = \{A^C | A \in N\}$. In this case, A^C is not a complement operator, but simply the name of the new nonterminal symbol in N^C.

Definition 5.4 (Extended Complement Operator). Given a string (sentential form) over the alphabet $\Gamma^C = (\Sigma \cup N \cup N^C)$, and a complement relation ρ on Σ we extend the definition of the complement operator in the following way:

- $\epsilon^c = \epsilon$
- $(aw)^c = \rho(a)w^c$, if $w \in (\Gamma^C)^*$ and $a \in \Sigma$
- $(Aw)^c = A^C w^r$, if $w \in (\Gamma^C)^*$ and $A \in N$
- $(A^C w)^c = Aw^r$, if $w \in (\Gamma^C)^*$ and $A^C \in N^C$

Definition 5.5 (CCFG). A CFG with complement operator (CCFG) is a tuple $G = (N, \Sigma, P, S, \rho)$, where:

- N is a finite set nonterminals;
- Σ is the alphabet of terminal symbols;
- P is a set of productions rules of the form $A \to \alpha$, where:
  - $A \in N$ is a nonterminal;
  - $\alpha \in (\Sigma \cup N \cup N^C)^*$ is a sentential form;
- $S \in N$ is the initial symbol.
- ρ is a complement relation on Σ.

Definition 5.6 (One-step derivation relation on $(\Gamma^C)^*$). Given a CCFG $G = (N, \Sigma, P, S, \rho)$, we say that $w_1 \Rightarrow_G w_2$ when:

- $w_1 = \gamma_1 A \gamma_2$, $w_2 = \gamma_1 \alpha \gamma_2$, $A \to \alpha \in P$, $\gamma_1, \gamma_2, \alpha \in (\Gamma^C)^*$
- $w_1 = \gamma_1 A^C \gamma_2$, $w_2 = \gamma_1 \alpha^c \gamma_2$, $A \to \alpha \in P$, $\gamma_1, \gamma_2, \alpha \in (\Gamma^C)^*$

Definition 5.7 (Multiple steps derivation relation on $(\Gamma^C)^*$). $\Rightarrow_G^*$ is the reflexive transitive closure of $\Rightarrow_G$.

When the grammar is clear from the context we write simply $\Rightarrow$ for the one-step derivation relation and $\Rightarrow^*$ for the multiple steps derivation relation.

Definition 5.8 ($L(G)$)**.** Language generated by a CCFG $G = (N, \Sigma, P, S, \rho)$:
$L(G) = \{\omega \in \Sigma^* \mid S \Rightarrow^* \omega\}$

Theorem 5.1. *Given a CCFG $G = (N, \Sigma, P, S)$, $A \Rightarrow_G^* \omega$ if and only if $A^C \Rightarrow_G^* \omega^c$.*

Proof. The theorem follows trivially from the definition of $\Rightarrow_G$. □

Like the reverse operator, the complement operator can be interpreted as implicit new rules, therefore the complement operator does not increase the power of CFG grammars. The presence of complement operators may though increase the compression power of a grammar.

6 Augmented Grammars

An Augmented Grammar is a CFG augmented with both reverse and complement operators. To model the behavior of these operators working together we chose to understand that there are two independent labels (C and R) in the nonterminals.

Definition 6.1 (N^{CR})**.** Given a set of nonterminal symbols, N, we define a set of new nonterminals $N^{CR} = \{A^{CR} | A \in N\}$. In this case, A^{CR} is simply the name of the new nonterminal symbol in N^{CR}.

Then we extend Reverse and Complement definitions to strings (sentential forms) over the alphabet $(\Sigma \cup N \cup N^C \cup N^R \cup N^{CR})$

Definition 6.2 (Complement)**.** Given a string (sentential form) over the alphabet $\Gamma^{CR} = (\Sigma \cup N \cup N^C \cup N^R \cup N^{CR})$, and a complement relation ρ on Σ we extend the definition of the reverse operator in the following way:

- $\epsilon^c = \epsilon$
- $(aw)^c = \rho(a)w^c$, if $w \in (\Gamma^{CR})^*$ and $a \in \Sigma$
- $(Aw)^c = A^C w^r$, if $w \in (\Gamma^{CR})^*$ and $A \in N$
- $(A^R w)^c = A^{CR} w^r$, if $w \in (\Gamma^{CR})^*$ and $A \in N^R$
- $(A^C w)^c = Aw^r$, if $w \in (\Gamma^{CR})^*$ and $A^C \in N^C$
- $(A^{CR} w)^c = A^R w^r$, if $w \in (\Gamma^{CR})^*$ and $A^{CR} \in N^{CR}$

Definition 6.3 (Reverse)**.** Given a string (sentential form) over the alphabet $\Gamma^{CR} = (\Sigma \cup N \cup N^C \cup N^R \cup N^{CR})$, we extend the definition of the reverse operator in the following way:

- $\epsilon^r = \epsilon$
- $(aw)^r = w^r a$, if $w \in (\Gamma^{CR})^*$ and $a \in \Sigma$
- $(Aw)^r = w^r A^R$, if $w \in (\Gamma^{CR})^*$ and $A \in N$

- $(A^C w)^r = w^r A^{CR}$, if $w \in (\Gamma^{CR})^*$ and $A^C \in N^C$
- $(A^R w)^r = w^r A$, if $w \in (\Gamma^{CR})^*$ and $A^R \in N^R$
- $(A^{CR} w)^r = w^r A^C$, if $w \in (\Gamma^{CR})^*$ and $A^{CR} \in N^{CR}$

Definition 6.4 (ACFG)**.** An Augmentd Context-Free Grammar (ACFG) is a tuple $G = (N, \Sigma, P, S, \rho)$, where:

- N is a finite set nonterminals;
- Σ is the alphabet of terminal symbols;
- P is a set of productions rules of the form $A \to \alpha$, where:
  - $A \in N$ is a nonterminal;
  - $\alpha \in (\Sigma \cup N \cup N^C \cup N^R \cup N^{CR})^*$ is a sentential form;
- $S \in N$ is the initial symbol.
- ρ is a complement relation on Σ.

Definition 6.5 (One-step derivation relation on $(\Gamma^{CR})^*$)**.** Given an ACFG $G = (N, \Sigma, P, S, \rho)$, we say that $w_1 \Rightarrow_G w_2$ when:

- $w_1 = \gamma_1 A \gamma_2$, $w_2 = \gamma_1 \alpha \gamma_2$, $A \to \alpha \in P$, $\gamma_1, \gamma_2, \alpha \in (\Gamma^{CR})^*$
- $w_1 = \gamma_1 A^R \gamma_2$, $w_2 = \gamma_1 \alpha^r \gamma_2$, $A \to \alpha \in P$, $\gamma_1, \gamma_2, \alpha \in (\Gamma^{CR})^*$
- $w_1 = \gamma_1 A^C \gamma_2$, $w_2 = \gamma_1 \alpha^c \gamma_2$, $A \to \alpha \in P$, $\gamma_1, \gamma_2, \alpha \in (\Gamma^{CR})^*$
- $w_1 = \gamma_1 A^{CR} \gamma_2$, $w_2 = \gamma_1 (\alpha^c)^r \gamma_2$, $A \to \alpha \in P$, $\gamma_1, \gamma_2, \alpha \in (\Gamma^{CR})^*$

Definition 6.6 (Multiple steps derivation relation on $(\Gamma^{CR})^*$)**.** $\Rightarrow_G^*$ is the reflexive transitive closure of $\Rightarrow_G$.

When the grammar is clear from the context we write simply $\Rightarrow$ for the one-step derivation relation and $\Rightarrow^*$ for the multiple steps derivation relation.

Definition 6.7 ($L(G)$)**.** Language generated by an ACFG $G = (N, \Sigma, P, S, \rho)$:
$L(G) = \{\omega \in \Sigma^* \mid S \Rightarrow^* \omega\}$

Theorem 6.1. *Given an ACFG $G = (N, \Sigma, P, S, \rho)$, then:*

- *$A \Rightarrow_G^* \omega$ if and only if $A^C \Rightarrow_G^* \omega^c$.*
- *$A \Rightarrow_G^* \omega$ if and only if $A^R \Rightarrow_G^* \omega^r$.*
- *$A \Rightarrow_G^* \omega$ if and only if $A^{CR} \Rightarrow_G^* (\omega^c)^r$.*

Proof. The theorem follows trivially from the definition of $\Rightarrow_G$. □

In the case of CFGs, we consider each nonterminal as having unitary length, following the same idea, to compute the size of an ACFG we consider each nonterminal in $(N \cup N^C \cup N^R \cup N^{CR})$ to have one unit length.

Definition 6.8 ($|G|$)**.** Given an ACFG $G = (N, \Sigma, P, S, \rho)$, we define $|G| = \sum_{A \to \alpha \in P}(|\alpha|)$

```
while (c=getchar()!=EOF) {
    append c to the RHS of the rule $S \to \alpha$.
    while there is a digram d occuring twice {
    // both appending c and replacing d by $N_d$
    // may create a digram occuring twice
        if (other occurrence is the RHS of $N_d \to d$) {
            replace d by $N_d$
        } else {
            create a new rule $N_d \to d$
            replace both d occurrences by $N_d$
        }
        if (there is a rule r used only once) {
            expand r
        }
    }
}
```

Table 1: Sequitur Algorithm.

7 Heuristics and their ACFG generalization

The goal of this section is to argue that very little modification is needed in the Sequitur or IRR algorithms to adapt them to ACFGs.

7.1 Sequitur

Sequitur [18] is a linear time heuristics to the minimal grammar problem. A high-level description of the algorithm is given in table 1.

Sequitur can be implemented in O(n) time by representing the RHS of each rule as a list of symbols in $(\Sigma \cup N)$ and keeping a hashtable with a pointer for the occurrence of each digram.

To adapt the algorithm to ACFGs, you can simply choose a canonical digram between w, w^r, w^c and w^{cr}, for example, the first in alphabetical order, and always use the canonical digram when accessing the hashtable, besides you should observe the right choice between N_d, N_d^R, N_d^C and N_d^{CR} when replacing the digram d.

7.2 IRR

Table 2 shows IRR algorithm, described in [6], into which most offline algorithms for minimal grammars fit. In the following description, $repeats(P)$ denotes all strings with two or more non-overlapping occurrences in the RHS of P rules, $f(w, P)$ is a scoring function to chose the "best" string according to some heuristic, and $P_{w \to N}$ is the set of grammar rules produced by replacing all occurrences of w by N in the RHS of the set of rules P.

Common choices for the function $f(w, P)$ are:

- $f(w, P) = |w|$. This heuristic will substitute the longest repeating string first. Following [6] we will call IRR algorithm with this function IRR-ML. It is used in LFS2

```
IRR(s)
begin
    $N \leftarrow \{S\}$
    $P \leftarrow \{S \rightarrow s\}$
    while $(\exists w : w \leftarrow argmax_{\alpha \in repeats(P)} f(\alpha, P))$ and $(|P_{w \rightarrow N_w}| < |P|)$
        $N \leftarrow N \cup \{N_w\}$
        $P \leftarrow P_{w \rightarrow N_w} \cup \{N_w \rightarrow w\}$
    end-while
end
```

Table 2: IRR Algorithm.

[17] other works such as [5] use similar ideas but do not search for string matches in the newly generated rules.

- $f(w, P) = \#occurrences(w)$. This heuristic will substitute the string with most repetitions first. Following [6] we will call IRR algorithm with this function IRR-MO. The implementation in this work uses a generalized suffix tree for finding the set $repeats(P)$, therefore all w are maximal substrings with a given number of repetitions. This is the same idea used by the Re-Pair algorithm as analyzed in [7], whereas the original Re-Pair description in [15] works with strings of length 2.

- $f(w, P) = |P| - |P_{w \rightarrow N_w}|$. This heuristic will substitute the string which will cause the most reduction in the grammar size. This function is used in [19]. Experiments in [19] and [6] show that this is heuristic finds smaller grammars than the first two. Following [6] we will call IRR algorithm with this function IRR-MC.

To adapt the code in table 2 to deal with ACFGs, only two changes are needed:

- $repeats(P)$ will look for repeated occurrences of either w, w^r, w^c or w^{cr}. In the current implementation this is achieved by creating a generalizes suffix tree (GST) [11] into which we append $\alpha, \alpha^r, \alpha^c$ and α^{cr}, for all $A \rightarrow \alpha \in P$. Commons substrings in this GST go into $repeats(P)$.

- $P \leftarrow P_{w \rightarrow N_w} \cup \{N_w \rightarrow w\}$. In this case we choose a canonical RHS for the grammar rules $\{N_w \rightarrow w\}$ as the first in alphabetical order between w, w^r, w^c and w^{cr}, then $P_{w \rightarrow N_w}$ will change each ocurrence of w, w^r, w^c and w^{cr} by either N_w, N_w^R, N_w^C or N_w^{CR}, whichever yields the sequence being replaced.

8 Experiments

8.1 Genome sequences

In this section, we present the results of applying several minimal CFG heuristics to seven different virus genome files. Standard CFG heuristics are compared to their CCFG, RCFG, and ACFG generalizations. File properties are described in table 3.

Virus	File Size (bytes)	Baltimore Group	Accession Number
HPV-60	7313	G1	NC_001693.1
Parvovirus-h1	5176	G2	X01457.1
Rotavirus-A	18572	G3	NC_011507.2
SARS-Cov-2	29903	G4	NC_045512.2
Ebola	19959	G5	AF086833.2
HIV-1	9003	G6	MN692147.1
Hepatitis-B	3125	G7	AF384372.1

Table 3: Virus genome file details.

Virus	CFG RGS	RCFG RGS	CCFG RGS	ACFG RGS
HPV-60	0,413	0,393	0,389	0,365
Parvovirus-h1	0,426	0,405	0,396	0,370
Rotavirus-A	0,373	0,354	0,352	0,333
SARS-Cov-2	0,360	0,341	0,340	0,321
Ebola	0,361	0,341	0,339	0,323
HIV-1	0,404	0,382	0,384	0,363
Hepatitis-B	0,467	0,434	0,429	0,412
Mean	0,401	0,379	0,376	0,355

Table 4: IRR-ML RGS for virus genomes.

Table 5 presents RGS values obtained from Sequitur heuristics, whereas table 4 presentsthe same values for the IRR-ML heuristics and finally table 6 presents values for the IRR-MC heuristics.

The results corroborate previous experimental results [6][19] that show IRR-MC as better than Sequitur or other IRR variants. Furthermore, the results suggest that CCFG heuristics perform better than their usual CFG counterpart.

We highlight that the complement relation used was the Watson–Crick complement relation, hence each digraph has a different complement, therefore the number of potential size 2 rules is 8, whereas for the reverse relation it is 10 because strings such as "AA" have the complement identical to the original string. So CCFGs may perform better than RCFGs, especially on small alphabets. Surprisingly ACFGs seem not to improve RCFGs performance.

8.2 Random Files compared to genome sequences

Several previous works on the smallest grammar problem claim to detect an underlying "structure" on the input files by means of producing a grammar shorter than the input [18][6][20]. Notwithstanding these claims, one should expect "structure" (rule) detection in any random string (no matter its nature), because in an alphabet of size n there are $O(n^2)$ digraphs. In this section, we aim at evaluating smallest CCFG algorithms applied to genome sequences and compare the results to random sequences over a four-symbol

Virus	**CFG RGS**	**RCFG RGS**	**CCFG RGS**	**ACFG RGS**
HPV-60	0.341	0.333	0.307	0.317
Parvovirus-h1	0.363	0.341	0.322	0.326
Rotavirus-A	0.293	0.285	0.270	0.275
SARS-Cov-2	0.277	0.271	0.258	0.258
Ebola	0.285	0.275	0.263	0.263
HIV-1	0.328	0.312	0.300	0.298
Hepatitis-B	0.392	0.382	0.364	0.359
Mean	0.325	0.314	0.298	0.299

Table 5: Sequitur RGS for virus genomes.

Virus	**CFG RGS**	**RCFG RGS**	**CCFG RGS**	**ACFG RGS**
HPV-60	0,332	0,327	0,305	0,303
Parvovirus-h1	0,348	0,347	0,318	0,319
Rotavirus-A	0,287	0,287	0,269	0,267
SARS-Cov-2	0,274	0,273	0,255	0,260
Ebola	0,280	0,279	0,260	0,264
HIV-1	0,316	0,317	0,297	0,301
Hepatitis-B	0,386	0,379	0,347	0,354
Mean	0,318	0,316	0,293	0,295

Table 6: IRR-MC RGS for virus genomes.

Virus	$\|G\|_v$ **for CFG from Virus file**	$\|G\|_r$ **for CFG from same size random file**	$\|G\|_v/\|G\|_r$
HPV-60	2428	2504	0,970
Parvovirus-h1	1799	1876	0,959
Rotavirus-A	5338	5630	0,948
SARS-Cov-2	8206	8538	0,961
Ebola	5590	5972	0,936
HIV-1	2843	2986	0,952
Hepatitis-B	1206	1229	0,981
Mean	3916	4105	0,958

Table 7: IRR-MC CFG size for virus genomes and same size random files.

Virus	$\|G\|_v$ **for CCFG from Virus file**	$\|G\|_r$ **for CCFG from same size random file**	$\|G\|_v/\|G\|_r$
HPV-60	2230	2289	0,974
Parvovirus-h1	1648	1698	0,971
Rotavirus-A	4988	5178	0,963
SARS-Cov-2	7633	7868	0,970
Ebola	5188	5524	0,939
HIV-1	2678	2741	0,977
Hepatitis-B	1084	1100	0,985
Mean	3636	3771	0,969

Table 8: IRR-MC CCFG size for virus genomes and same size random files.

alphabet, to check if there is any gain over the random sequences, which could be attributed to some "structure" due to the nature of the data.

Table 7 displays grammar size ($|G|$) obtained from IRR-MC algorithm for the virus files and compared to ramdom generated files of the same size. Table 8 displays the same information obtained from IRR-MC variation for CCFGs.

Tables 7 and 8 show that virus file generate a slightly smaller grammar (4.2% and 3.1% respectively). Tests were also performed with the Sequitur algorithm and its CCFG variation, achieving similar results (3.0% and 2.8% respectively). This small difference may come from genome structure, possibly from sequence repetition, though other quirks, such as codon usage bias [3], may explain it.

8.3 Size of alphabet

This section addresses a rather neglected subject in the minimal grammar literature, namely the influence of the size of the alphabet on the grammar size. Given an alphabet with n symbols, there are at most n^2 digraphs and at most $n^2/2$ digraphs modulo a complement relation with no identity. This means that we expected a smaller RGS as alphabet size shrinks, leading to more digraph repetitions. Furthermore, due to the quadratic number of digraphs, we expect similar RGS values if the alphabet size reduces by a factor of two and the file size reduces by a factor of four. Both hypotheses are confirmed by the experiments, as shown in figure 1.

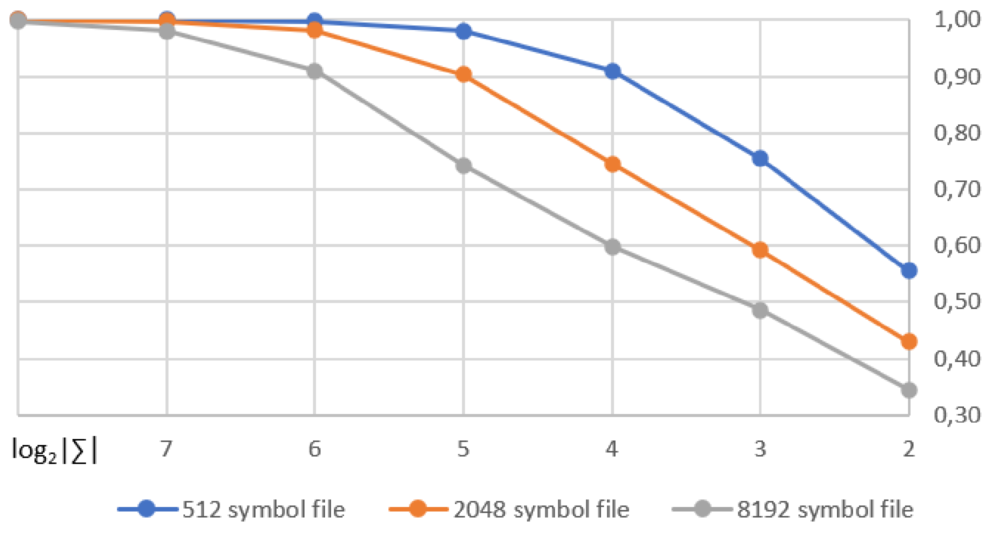

Figure 1: Relative Grammar Size as function of alphabet size

9 Conclusions and further works

The smallest grammar problem is an extensively studied problem. It consists of finding the smallest CFG that generates only a given string. In this paper, we proposed two extensions of CFGs: reverse operator and complement operator, which lead to three extensions of CFGs:

- RCFGs. CFGs extended with reverse operator.
- CCFGs. CFGs extended with complement operator.
- ACFGs. CFGs extended with both reverse and complement operators.

We argued that the heuristics used to find approximate solutions to the smallest grammar problem can be easily adapted to these extensions. We performed experiments with several virus genomes, as well as with randomly generated files, leading to the following conclusions:

- Both operators performed better (individually) than plain CFG.
- The complement operator always performed better than the reverse operator.
- The complement operator alone performed better than both operators together in all heuristics but IRR-ML.

- Virus Genomes generate a 4.2% smaller CFG and a 3.1% smaller CCFG than a randomly generated file of the same size.
- Alphabet size play a huge role in the size of the resulting grammar, the bigger the Alphabet the bigger a file must be to achieve the same Relative Grammar Size.

As further works we intend to:

- Apply the generalized heuristics developed in this work to text compression, proof compression [10], and other kinds of data.
- Investigate the generalization of the heuristics for the Smallest Grammar Problem to a broader class of Grammars, such as indexed grammars or linear indexed grammars.
- Investigate the nature of the structure detected by a grammar produced from an English text.

References

[1] A. Apostolico and S. Lonardi. Off-line compression by greedy textual substitution. *Proceedings of the IEEE*, 88(11):1733–1744, November 2000.

[2] Alberto Apostolico and Stefano Lonardi. Some theory and practice of greedy off-line textual substitution. In *Data Compression Conference, DCC 1998, Snowbird, Utah, USA*, pages 119–128. IEEE Computer Society, 1998.

[3] John Athey, Aikaterini Alexaki, Ekaterina Osipova, Alexandre Rostovtsev, Luis V. Santana-Quintero, Upendra Katneni, Vahan Simonyan, and Chava Kimchi-Sarfaty. A new and updated resource for codon usage tables. *BMC Bioinformatics*, 18(1):391, Sep 2017.

[4] Birner B. *The discourse function of inversion in English. Outstanding Dissertations in Linguistics.* Routledge, New York, 1996.

[5] J. Bentley and D. McIlroy. Data compression using long common strings. In *Proceedings DCC'99 Data Compression Conference*, pages 287–295, 1999.

[6] Rafael Carrascosa, François Coste, Matthias Gallé, and Gabriel G. Infante López. Choosing Word Occurrences for the Smallest Grammar Problem. In *Proceedings of the Fourth International Conference on Language and Automata Theory and Applications*, volume 6031 of *Lecture Notes in Computer Science*, pages 154–165. Springer International Publishing, 2010.

[7] Moses Charikar, Eric Lehman, Ding Liu, Rina Panigrahy, Manoj Prabhakaran, Amit Sahai, and Abhi Shelat. The smallest grammar problem. *IEEE Trans. Inf. Theory*, 51(7):2554–2576, 2005.

[8] Carl G. de Marcken. *Unsupervised language acquisition.* PhD thesis, MIT, 1996.

[9] Scott C Evans, Antonis Kourtidis, T Stephen Markham, Jonathan Miller, Douglas S Conklin, and Andrew S Torres. MicroRNA target detection and analysis for genes related to breast cancer using MDLcompress. *EURASIP Journal on Bioinformatics and Systems Biology*, 2007(1):43670, 2007.

[10] L. Gordeev, E. H. Haeusler, and V. G. da Costa. Proof compressions with circuit-structured substitutions. *Journal of Mathematical Sciences*, 158(5):645–658, May 2009.

[11] Dan Gusfield. *Algorithms on Strings, Trees, and Sequences: Computer Science and Computational Biology*. Cambridge University Press, New York, 1997.

[12] John E. Hopcroft, Rajeev Motwani, and Jeffrey D. Ullman. *Introduction to Automata Theory, Languages, and Computation (3rd Edition)*. Addison-Wesley Longman Publishing Co., Inc., USA, 2006.

[13] En hui Yang and John C. Kieffer. Efficient universal lossless data compression algorithms based on a greedy sequential grammar transform – part one: Without context models. *IEEE TRANSACTIONS ON INFORMATION THEORY*, 46(3):755–777, 2000.

[14] John C. Kieffer and En-Hui Yang. Grammar-based codes: A new class of universal lossless source codes. *IEEE Trans. Inf. Theory*, 46(3):737–754, 2000.

[15] Jesper Larsson and Alistair Moffat. Offine dictionary-based compression. In *Proc. IEEE*, pages 296–305. IEEE Computer Society, 1999.

[16] M. Lohrey. Algorithmics on slp-compressed strings: A survey. *Groups Complexity Cryptology*, 4(2):241–299, 2012.

[17] Ryosuke Nakamura, Shunsuke Inenaga, Hideo Bannai, Takashi Funamoto, Masayuki Takeda, and Ayumi Shinohara. Linear-time text compression by longest-first substitution. *Algorithms*, 2(4):1429–1448, 2009.

[18] Craig Nevill-Manning and Ian Witten. Identifying hierarchical structure in sequences: A linear-time algorithm. *Journal of Artificial Intelligence*, 7:67–82, 1997.

[19] Craig Nevill-Manning and Ian Witten. Online and offline heuristics for inferring hierarchies of repetitions in sentences. In *Data Compression Conference*, pages 1745–1755, Los Alamitos, 2000. IEEE.

[20] Craig G. Nevill-Manning. *Inferring Sequential Structure*. PhD thesis, University of Waikato, 1996.

[21] Lisa G. Shaffer R.J.M McKinlay Gardner, Grant R. Sutherland. *Chromosome Abnormalities and Genetic Counseling* . Oxford University Press, New York, 2011.