On the Complexity of Grammar-Based Compression over Fixed Alphabets

Katrin Casel*[1], Henning Fernau[2], Serge Gaspers†[3], Benjamin Gras[4], and Markus L. Schmid[5]

1 **Trier University, Fachbereich IV – Abteilung Informatikwissenschaften, Trier, Germany**
Casel@uni-trier.de
2 **Trier University, Fachbereich IV – Abteilung Informatikwissenschaften, Trier, Germany**
Fernau@uni-trier.de
3 **UNSW Australia, Sydney, Australia, and Data61 (formerly: NICTA), CSIRO, Sydney, Australia**
sergeg@cse.unsw.edu.au
4 **École Normale Superieure de Lyon, Département Informatique, Lyon, France**
benjamin.gras@ens-lyon.fr
5 **Trier University, Fachbereich IV – Abteilung Informatikwissenschaften, Trier, Germany**
MSchmid@uni-trier.de

Abstract

It is shown that the shortest-grammar problem remains NP-complete if the alphabet is fixed and has a size of at least 24 (which settles an open question). On the other hand, this problem can be solved in polynomial-time, if the number of nonterminals is bounded, which is shown by encoding the problem as a problem on graphs with interval structure. Furthermore, we present an $\mathcal{O}(3^n)$ exact exponential-time algorithm, based on dynamic programming. Similar results are also given for 1-level grammars, i. e., grammars for which only the start rule contains nonterminals on the right side (thus, investigating the impact of the "hierarchical depth" on the complexity of the shortest-grammar problem).

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems, E.4 Coding and Information Theory

Keywords and phrases Grammar-Based Compression, Straight-Line Programs, NP-Completeness, Exact Exponential Time Algorithms

Digital Object Identifier 10.4230/LIPIcs.ICALP.2016.122

1 Introduction

While in the early days of computer science, the most important requirements for compression schemes were fast (i. e., linear or near linear time) compression and decompression, nowadays their investigation regarding whether they are suitable for solving problems directly on

* Katrin Casel is supported by the Deutsche Forschungsgemeinschaft (FE 560/6-1).
† Serge Gaspers is the recipient of an Australian Research Council (ARC) Future Fellowship (FT140100048) and acknowledges support under the ARC's Discovery Projects funding scheme (DP150101134). NICTA is funded by the Australian Government through the Department of Communications and the ARC through the ICT Centre of Excellence Program.

43rd International Colloquium on Automata, Languages, and Programming (ICALP 2016).
Editors: Ioannis Chatzigiannakis, Michael Mitzenmacher, Yuval Rabani, and Davide Sangiorgi;
Article No. 122; pp. 122:1–122:14

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

the compressed data without prior decompression forms a vibrant research area, usually subsumed under the term *algorithmics on compressed strings*.

Compressing a word by a context-free grammar, so-called *grammar-based compression*, is particularly well suited for this purpose.[1] Nevill-Manning and Witten [17, 16], and Kieffer et al. [10, 9, 23] are stated as the origins of this concept, but a closer look into the older literature reveals that the *external pointer macro* (*without overlapping and with pointer size* 1) defined by Storer and Szymanski [22, 21] is also equivalent to grammar-based compression.

The success of grammars with respect to algorithmics on compressed strings is due to the fact that they cover many compression schemes from practice (most notably, the family of Lempel-Ziv encodings) and that they are mathematically easy to handle (see Lohrey [11] for a survey on the role of grammar-based compression for algorithmics on compressed strings). Many basic problems on strings, e. g., comparison, pattern matching, membership in a regular language, retrieving subwords, etc. can all be solved in polynomial-time directly on the grammars [11]. In addition, grammar-based compression has been successfully applied in combinatorial group theory and to prove problems in computational topology to be polynomial-time solvable [11]. Grammars as compressions have also been extended to more complicated objects, e. g., trees (see [1, 12, 13, 14]) and two-dimensional words (see [3]).

On the other hand, work on the *shortest-grammar problem*, i. e., computing a minimal grammar for a given word w, is somewhat scarce; its NP-completeness, the major downside of grammar-based compression, has been used as a justification to focus on approximation algorithms. In this regard, the best achieved approximation ratio is $\mathcal{O}(\log(\frac{|w|}{m^*}))$ (see [18, 4]), where m^* is the size of a smallest grammar, and an approximation ratio better than $\frac{8569}{8568} \approx 1.0001$ is not possible (see [4]), assuming $\mathsf{P} \neq \mathsf{NP}$ (the research seems to have stagnated at this gap between lower and upper bound). However, the existing hardness reductions (also for the inapproximability result) have a serious deficiency: they assume the terminal alphabet to be unbounded. In [2], it is claimed that the hardness for alphabets of size 3 follows from [21], but a closer look into [21] does not confirm this. For string problems, where we typically deal with not only constant, but also very small alphabets, e. g., of size 2 or 4, this is rather unsatisfying. Another neglected aspect is the parameterised point of view, i. e., can minimal grammars be efficiently computed, if certain parameters (e. g., alphabet size, levels of the derivation tree, number of rules) are bounded? Furthermore, the fact that for grammar compressions basic problems can be solved without decompression motivates scenarios, where an extensive running time is invested only once, in order to obtain an optimal compression, which is then stored and worked with. This assumption naturally leads to exact exponential-time algorithms, which are not yet considered in the literature.

We investigate these aspects of the shortest-grammar problem. First, we close the gap with respect to fixed alphabets left open in the literature, by showing the NP-completeness for alphabets of size 24 in Section 3, which provides a more solid foundation for approximations (or heuristics), but leaves the cases of small alphabets open.[2] After this negative result, in Section 4, we show that minimal grammars can be computed in polynomial-time, provided that the size of the nonterminal alphabet is bounded. This is achieved by a reduction to a graph problem, which, since the graphs are structurally simple, can be efficiently solved (note that especially for string problems successful applications of this common algorithmic technique are rare). Additionally, we show that an FPT-algorithm with respect to this

[1] Such context-free grammars are also called *straight-line programs* in the literature.
[2] Note that this negative result also transfers to the shortest-grammar problem for trees.

parameter is unlikely (under complexity theoretical assumptions). Finally, we turn our attention to exact exponential-time algorithms in Section 5 and first observe that brute-force algorithms with running time $\mathcal{O}(c^{|w|})$, for a constant c, can be easily found, but we also present an $\mathcal{O}(3^{|w|})$ dynamic programming algorithm.

Moreover, these questions are also investigated for 1-*level grammars*, i. e., only the start rule contains nonterminals; thus, measuring the impact of the "hierarchical depth" of the grammars. Considering that the exploitation of hierarchical structure is one of the main features of grammars (allowing exponential compression rates, in contrast to 1-level grammars, where quadratic is the best), it is surprising that our results suggest that computing general grammars is, if at all, only insignificantly more difficult than computing 1-level grammars.

Due to space restriction, we only give proof sketches for the main results.

2 Preliminaries

Let $\mathbb{N} = \{1, 2, 3, \ldots\}$. By $|A|$, we denote the cardinality of a set A. Let Σ be a finite alphabet of *symbols*. A *word* or *string* (over Σ) is a sequence of symbols from Σ. For any word w over Σ, $|w|$ denotes the length of w and ε denotes the *empty word*, i. e., $|\varepsilon| = 0$. The symbol Σ^+ denotes the set of all non-empty words over Σ and $\Sigma^* = \Sigma^+ \cup \{\varepsilon\}$. For the *concatenation* of two words w_1, w_2 we write $w_1 \cdot w_2$ or simply $w_1 w_2$. For every symbol $a \in \Sigma$, by $|w|_a$ we denote the number of occurrences of symbol a in w. We say that a word $v \in \Sigma^*$ is a *factor* of a word $w \in \Sigma^*$ if there are $u_1, u_2 \in \Sigma^*$ such that $w = u_1 v u_2$. If $u_1 = \varepsilon$ or $u_2 = \varepsilon$, then v is a *prefix* (or a *suffix*, respectively) of w. Furthermore, $\mathsf{F}(w) = \{u \colon u \text{ is a factor of } w\}$ and $\mathsf{F}_{\geq 2}(w) = \{u \colon u \in \mathsf{F}(w), |u| \geq 2\}$. For a position j, $1 \leq j \leq |w|$, we refer to the symbol at position j of w by the expression $w[j]$ and $w[j..j'] = w[j]w[j+1] \ldots w[j']$, $j \leq j' \leq |w|$. By w^R, we denote the *reversal* of w, i. e., $w^R = w[n]w[n-1] \ldots w[1]$, where $|w| = n$.

A *factorisation* of a word w is a tuple $(u_1, u_2, \ldots, u_k)$ with $u_i \neq \varepsilon$, $1 \leq i \leq k$ such that $w = u_1 u_2 \ldots u_k$. A factorisation $p = (u_1, u_2, \ldots, u_k)$ is a *refinement* of a factorisation $q = (v_1, v_2, \ldots, v_m)$, denoted by $p \preceq q$, if $(u_{j_{i-1}+1}, u_{j_{i-1}+2}, \ldots, u_{j_i})$ is a factorisation of v_i, $1 \leq i \leq m$ for some $\{j_i\}_{0 \leq i \leq m}$, with $0 = j_0 < j_1 < \ldots < j_m = k$.

Grammars: A *context-free grammar* is a tuple $G = (N, \Sigma, R, S)$, where N is the set of *nonterminals*, Σ is the *terminal alphabet*, $S \in N$ is the *start symbol* and $R \subseteq N \times (N \cup \Sigma)^+$ is the set of *rules* (as a convention, we write rules $(A, w) \in R$ also in the form $A \to w$). A context-free grammar $G = (N, \Sigma, R, S)$ is a *singleton grammar* if R is a total function $N \to (N \cup \Sigma)^+$ and the relation $\{(A, B) \colon (A, w) \in R, |\alpha|_B \geq 1\}$ is acyclic.

For a singleton grammar $G = (N, \Sigma, R, S)$, let $\mathsf{D}_G : (N \cup \Sigma) \to (N \cup \Sigma)^+$ be defined by $\mathsf{D}_G(A) = R(A)$, $A \in N$, and $\mathsf{D}_G(a) = a$, $a \in \Sigma$. We extend D_G to a morphism $(N \cup \Sigma)^+ \to (N \cup \Sigma)^+$ by setting $\mathsf{D}_G(\alpha_1 \alpha_2 \ldots \alpha_n) = \mathsf{D}_G(\alpha_1)\,\mathsf{D}_G(\alpha_2) \ldots \mathsf{D}_G(\alpha_n)$, for $\alpha_i \in (N \cup \Sigma)$, $1 \leq i \leq n$. Furthermore, for every $\alpha \in (N \cup \Sigma)^+$, we set $\mathsf{D}^1_G(\alpha) = \mathsf{D}_G(\alpha)$, $\mathsf{D}^k_G(\alpha) = \mathsf{D}(\mathsf{D}^{k-1}_G(\alpha))$, for every $k \geq 2$, and $\mathfrak{D}_G(\alpha) = \lim_{k \to \infty} \mathsf{D}^k_G(\alpha)$ is the *derivative* of α. By definition, $\mathfrak{D}_G(\alpha)$ exists for every $\alpha \in (N \cup \Sigma)^+$ and is an element from Σ^+. The size of the singleton grammar G is defined by $|G| = \sum_{A \in N} |\,\mathsf{D}_G(A)|$ and its number of *levels* is $\min\{k \colon \mathsf{D}^k_G(S) = \mathfrak{D}_G(S)\}$. In particular, a grammar with d levels is a *d-level grammar*.

From now on, we simply use the term *grammar* instead of singleton grammar and if the grammar under consideration is clear from the context, we also drop the subscript G. We set $\mathfrak{D}(G) = \mathfrak{D}(S)$ and say that G *is a grammar for* $\mathfrak{D}(G)$. In the tuple (N, Σ, R, S), we sometimes replace S directly by $\mathsf{D}(S)$, which we then call the *compressed string* (*of* G) and which we denote by cs.

Let $G = (N, \Sigma, R, cs)$ be a 1-level grammar. The *profit* of a rule $(A, \alpha) \in R$ is defined by $\mathsf{p}(A) = |cs|_A(|\alpha| - 1) - |\alpha|$. Intuitively speaking, if all occurrences of A in cs are replaced by α and the rule $A \to \alpha$ is deleted, then the size of the grammar increases by exactly $\mathsf{p}(A)$. Consequently, $|G| = |\mathfrak{D}(G)| - \sum_{A \in N} \mathsf{p}(A)$.

A grammar G is *minimal* if $|G| = \min\{|G'| \colon G' \text{ is a grammar for } \mathfrak{D}(G)\}$ and the problem of computing small grammars is defined as follows:

Shortest Grammar Problem (SGP)
Instance: A word w and a $k \in \mathbb{N}$.
Question: Does there exist a grammar G with $\mathfrak{D}(G) = w$ and $|G| \leq k$?

The Shortest 1-Level Grammar Problem (1-SGP) is defined analogously, with the only difference that we ask for a 1-level grammar of size at most k.

Examples: Even for small – say binary – alphabets and a fixed word with a simple structure, finding minimal grammars can be surprisingly difficult. In order to substantiate this claim, let $w = \prod_{i=1}^{n} 10^i$ be a word over $\Sigma = \{0, 1\}$, where $n = 2^k$, $k \in \mathbb{N}$. One way of compressing w that comes to mind is by the use of rules $A_1 \to 10$, $A_i \to A_{i-1}0$, $2 \leq i \leq n-1$, and a compressed string $A_1 A_2 \dots A_{n-1} A_{n-1} 0$, which leads to a grammar G_1 of size $3n-1$. However, it is also possible to construct the factors 0^i, $1 \leq i \leq n$, "from the middle" by rules $A_1 \to 010$, $A_i \to 0A_{i-1}0$, $2 \leq i \leq \frac{n}{2} - 1$, and a compressed string $1(A_1)^2(A_2)^2 \dots$. By using these ideas, we can construct a smaller grammar G_2 of size $\frac{5n}{2} + 2k - 3$. Both of these grammars achieve a compression rate of order $\mathcal{O}(\sqrt{|w|})$, but, generally, grammars are capable of exponential compression rates (see [4]). Aiming for such exponential compression, it seems worthwhile to represent every unary factor 0^{2^ℓ}, $1 \leq \ell \leq k$, by a nonterminal B_ℓ (obviously, this requires only k rules of size 2) and then represent all unary factors by sums of these powers (e. g., 0^{74} is compressed by $B_1 B_3 B_6$). However, this yields a grammar G_3 of size $\frac{k(n+3)}{2} - 2$, which, if k is sufficiently large, is worse than the previous grammars.

A smaller grammar can be obtained by combining the idea of G_2 with that of representing factors 0^{2^ℓ} by nonterminals B_ℓ. More precisely, for every ℓ, $1 \leq i \leq k-2$, we represent 0^{2^ℓ} by an individual nonterminal B_ℓ and, in addition, we use rules $A_1 \to 010$, $A_i \to 0A_{i-1}0$, $2 \leq i \leq \frac{n}{4}$. Then the left and right half of w can be compressed in the way of G_2, with the only difference that in the right part, for every unary factor, we also need an occurrence of B_{k-1}, i. e., the compressed string is $1(A_1)^2 \dots (A_{\frac{n}{4}})^2 B_{k-2} (A_1 B_{k-1})^2 \dots (A_{\frac{n}{4}-1} B_{k-1})^2 A_{\frac{n}{4}} (B_{k-2})^3$. The size of this grammar G_4 is only $\frac{9n}{4} + 2k - 2$.

3 NP-Hardness of Computing Minimal Grammars for Fixed Alphabets

In [4], Charikar et al. prove the shortest-grammar problem to be NP-complete by a reduction from the vertex cover problem (which is based on ideas used by Storer and Szymanski in [22]). A simple modification of this reduction yields the following.

▶ **Theorem 1.** 1-SGP *is* NP*-complete.*

In these reductions, we encode the different vertices of a graph by single symbols and also use individual separator symbols (i. e., symbols with only one occurrence in the word to be compressed). This makes it particularly easy to devise suitable gadgets, but, on the other hand, it assumes that we have an arbitrarily large alphabet at our disposal, which, for practical situations, is not justified. In the remainder of this section, we shall extend these hardness results to the more realistic case of fixed alphabets. The general structure of

our reductions is similar to the ones of [4, 21], but, due to the constraint of having a fixed alphabet, they substantially differ on a more detailed level.

Since fixed alphabets make it impossible to use single symbols (or even words of constant size) as separators or as representing vertices, we need to use special encodings for which we are able to determine how a smallest grammar will compress them (in this regard, recall our examples from page 4 demonstrating how difficult it can be to determine a smallest grammar even for a single simply structured word). This constitutes a substantial technical challenge, which complicates our reductions considerably.

3.1 The 1-Level Case

As a tool for proving the hardness of 1-SGP, but also as a result in its own right, we first show that the compression of any 1-level grammar is at best quadratic (in contrast to general grammars, which can achieve exponential compression (see [4])).[3]

▶ **Lemma 2.** *Let G be a 1-level grammar. Then $|G| \geq 2\left\lfloor\sqrt{|\mathfrak{D}(G)|}\right\rfloor$.*

In order to prove the NP-hardness of 1-SGP for constant alphabets, we devise a reduction from the vertex cover problem. To this end, let $\mathcal{G} = (V, E)$ be a graph with $V = \{v_1, \ldots, v_n\}$ and $E = \{(v_{j_{2i-1}}, v_{j_{2i}}) \colon 1 \leq i \leq m\}$. Without loss of generality, we assume $n \geq 40$. We define $\Sigma = \{\mathtt{a}, \mathtt{b}, \diamond, \star, \#\}$ and $[\diamond] = \diamond^{n^3}$. For each i, $1 \leq i \leq n$, we encode v_i by a word $\overline{v_i} \in \{\mathtt{a}, \mathtt{b}\}^{\lceil\log(n)\rceil}$ such that $\overline{v_i} \neq \overline{v_j}$ if and only if $i \neq j$ (e. g., by taking $\overline{v_i}$ to be the binary representation of i over symbols $\mathtt{a}$ and $\mathtt{b}$ with $\lceil\log(n)\rceil$ many digits). We now define the following word over Σ:

$$w = \prod_{i=1}^{n} (\#\overline{v_i}[\diamond]\overline{v_i}\#[\diamond])^{2\lceil\log(n)\rceil+3} \prod_{i=1}^{n} (\#\overline{v_i}\#[\diamond])^{\lceil\log(n)\rceil+1} \prod_{i=1}^{m} (\#\overline{v_{j_{2i-1}}}\#\overline{v_{j_{2i}}}\#[\diamond])^2 \star [\diamond]^{n^3}.$$

▶ **Theorem 3.** 1-SGP *is* NP*-hard, even for* $|\Sigma| = 5$.

Proof Sketch. A smallest grammar for w produces the two parts to the left and right of $\star$ independently, since $|w|_\star = 1$. According to Lemma 2, the right side $[\diamond]^{n^3}$ is best compressed by n^3 occurrences of a nonterminal D with derivative $[\diamond]$ and, by a slightly more involved argument, it can be shown that also for the whole word w, it is still best to compress all occurrences of $[\diamond]$ by D. Having established this basic property, it is then possible to show that the remaining rules have derivative $\#\overline{v_i}$, $\overline{v_i}\#$ or $\#\overline{v_j}\#$. The grammar is smallest, if every edge $\#\overline{v_{j_{2i-1}}}\#\overline{v_{j_{2i}}}\#$ is compressed by using a rule of the last type; thus, those rules translate into a vertex cover. Analogously, a vertex cover translates into a grammar. ◀

3.2 The Multi-Level Case

In the above reduction, the main difficulty is the use of unary factors as separators. However, once those separators are in place, we know the factors of w that are produced by nonterminals and, for a minimal 1-level grammar, this already fully determines the compressed string and, thus, the grammar itself. For the multi-level case, the situation is much more complicated. Even if we manage to force the compressed string to factorise w into parts that are either separators or codewords of vertices, this only determines the top-most level of the grammar and we do not necessarily know how these single factors are further hierarchically compressed

[3] The bound of Lemma 2 is tight, e. g., consider $\mathtt{a}^{n^2}$ and a grammar with rules $S \to A^n$ and $A \to \mathtt{a}^n$.

and, more importantly, the dependencies between these compressions (i. e., how they share the same rules).

To deal with these issues, we rely on a larger alphabet Σ and we use palindromic codewords $u \star u^R$, where $\star \in \Sigma$ and u is a word over an alphabet of size 7 representing a 7-ary number. The purpose of the palindromic structure is twofold. Firstly, it implies that codewords always start and end with the same symbol, which, in the construction of w, makes it easier to avoid the situation that an overlapping between neighbouring codewords is repeated elsewhere in w (see Lemma 4). Secondly, if all codewords are produced by individual nonterminals, then we can show that they are produced best "from the middle", similar as the rules of the example grammar G_2 from page 4. In addition to this, we also need a vertex colouring and an edge colouring of certain variants of the graph to be encoded.

In order to formally define the reduction, we first give some preparatory definitions. Let $\Sigma = \{x_1, \ldots, x_7, d_1, \ldots, d_7, \star, \#, \mathfrak{c}_1, \mathfrak{c}_2, \$_1, \ldots, \$_6\}$ be an alphabet of size 24. The function $M\colon \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ is defined by $M(q,k) := \min\{r > 0\colon \exists t \in \mathbb{N}\colon q = tk + r\}$.[4] Let the functions $f\colon \mathbb{N} \to \{x_1, \ldots, x_7\}^+$ and $g\colon \mathbb{N} \to \{d_1, \ldots, d_7\}^+$ be defined by $f(q) := x_{a_0} x_{a_1} \ldots x_{a_k}$ and $g(q) := d_{a_0} d_{a_1} \ldots d_{a_k}$, for every $q \in \mathbb{N}$, where $k \in \mathbb{N} \cup \{0\}$ and $a_i \in \{1, 2, \ldots, 7\}$, $0 \leq i \leq k$, are such that $q = \sum_{i=0}^{k} a_i 7^i$ is satisfied.[5] For every $i \in \mathbb{N}$, let $\langle i \rangle_v := f(i) \star f(i)^R$ and $\langle i \rangle_\diamond := g(i) \star g(i)^R$. The factors $\langle i \rangle_v$ and $\langle i \rangle_\diamond$ are called *codewords*; $\langle i \rangle_v$ represents a vertex v_i, while the $\langle i \rangle_\diamond$ are used as separators. The functions f and g are bijections and they are 7-ary representations of the integers $n > 0$ (least significant digit first). Thus, for every $n, n' \in \mathbb{N}$ with $M(n,7) \neq M(n',7)$, the words $\langle n \rangle_v$ and $\langle n' \rangle_v$ do not share any prefixes or suffixes (and the same holds for the words $\langle n \rangle_\diamond$).

Let $\mathcal{G} = (V, E)$ be a subcubic graph (i. e., a graph with maximum degree 3) with $V = \{v_1, \ldots, v_n\}$ and $E = \{\{v_{j_{2i-1}}, v_{j_{2i}}\}\colon 1 \leq i \leq m\}$ (note that the vertex cover problem remains NP-hard if restricted to subcubic graphs (see [7])). Let $\mathcal{G}' = (V, E')$ be the multi-graph defined by $E' := \big\{\{v_{j_{2i}}, v_{j_{2i+1}}\}\colon 1 \leq i \leq m-1\big\}$. By [19], it is possible to compute in polynomial-time a proper edge-colouring (meaning a colouring such that no two edges which share one or two vertices have the same colour) for a multi-graph with at most $\lfloor \frac{3}{2} m \rfloor$ colours, where m is the maximum degree of the multi-graph. Since $\mathcal{G}$ is subcubic, the maximum degree of $\mathcal{G}'$ is three and we can compute a proper edge-colouring $C_e\colon E' \to \{1,2,3,4\}$ for $\mathcal{G}'$ with colours {1,2,3,4}. Let $\mathcal{G}^2 = (V, E'')$ be the graph defined by $E'' = \{\{u,v\}\colon \{u,w\}, \{w,v\} \in E \text{ for some } w \in V \setminus \{u,v\}, u \neq v\}$. Since $\mathcal{G}$ is subcubic, $\mathcal{G}^2$ has maximum degree at most six. Let $C_v\colon \{1, \ldots, n\} \to \{1,2,3,4,5,6,7\}$ be a proper vertex-colouring (defined over the vertex-indices of $V = \{v_1, \ldots, v_n\}$) for $\mathcal{G}^2$ with colours $\{1,2,3,4,5,6,7\}$. Such a colouring can be computed by an algorithmic version of Brook's theorem [20].

Let $w_\mathcal{G} = uvw$ be the word representing $\mathcal{G}$, where $u, v, w \in \Sigma^+$ are defined as follows.[6]

$$u = \prod_{j=0}^{6} \left(\prod_{i=1}^{14n} (\langle i \rangle_\diamond \ \langle M(i+j, 14n) \rangle_v) \right) \ \$_1$$

$$v = \prod_{i=1}^{n} (\# \ \langle 7i + C_v(i) \rangle_v \ \mathfrak{c}_1 \ \langle 7i - 1 \rangle_\diamond) \ \$_2 \prod_{i=1}^{n} (\# \ \langle 7i + C_v(i) \rangle_v \ \mathfrak{c}_2 \ \langle 7i - 2 \rangle_\diamond) \ \$_3$$

[4] M is the positive modulo-function, i. e., $M(q,k) = q \% k$, if $q \% k \neq 0$ and $M(q,k) = k$, otherwise.

[5] Since, for every $q \in \mathbb{N}$, there are unique $k \in \mathbb{N}$ and $a_i \in \{1, 2, \ldots, 7\}$, $1 \leq i \leq k$, such that $q = \sum_{i \geq 0}^{k} a_i 7^i$, the functions f and g are well-defined.

[6] Note that $m \leq \frac{3n}{2}$, so $7m < 14n$ in the word w.

$$\prod_{i=1}^{n}(\langle 7i+C_v(i)\rangle_v \ \#\ \langle 7i-2\rangle_\diamond\ \mathfrak{c}_1)\ \ \$_4 \prod_{i=1}^{n}(\langle 7i+C_v(i)\rangle_v \ \#\ \langle 7i-1\rangle_\diamond\ \mathfrak{c}_2)\ \ \$_5$$
$$\prod_{i=1}^{n}(\#\ \langle 7i+C_v(i)\rangle_v \ \#\ \langle 7i\rangle_\diamond)\ \ \$_6$$
$$w = \prod_{i=1}^{m-1}(\#\ \langle 7j_{2i-1}+C_v(j_{2i-1})\rangle_v \ \#\ \langle 7j_{2i}+C_v(j_{2i})\rangle_v \ \#\ \langle 7i+C_e(v_{j_{2i}},v_{j_{2i+1}})\rangle_\diamond)$$
$$\#\ \langle 7j_{2m-1}+C_v(j_{2m-1})\rangle_v \ \#\ \langle 7j_{2m}+C_v(j_{2m})\rangle_v \#$$

The next lemma states that any factor of $w_\mathcal{G}$ is not repeated, if it spans over the $\star$ of some codeword $\langle i\rangle_v$ or $\langle i\rangle_\diamond$ and also reaches over the boundaries of this codeword into some other factor. This property, which can be proven by a straightforward, but rather cumbersome analysis, is crucial for the correctness of the reduction and also responsible for the complicated structure of $w_\mathcal{G}$. Here, we only wish to point out that it follows from the fact that all occurrences of the same codeword are delimited by distinct symbols. This is ensured by the symbols $\#, \mathfrak{c}_1, \mathfrak{c}_2, \$_1, \ldots, \$_6$, by the fact that codewords $\langle i\rangle_v$ and $\langle i\rangle_\diamond$ start and end with $x_{M(i,7)}$ and $d_{M(i,7)}$, respectively, and, for the part w, by the colourings C_e and C_v.

▶ **Lemma 4.** *There is a minimal grammar $G = (N, \Sigma, R, S)$ for $w_\mathcal{G}$ such that, for every $A \in N$, $|\mathfrak{D}(A)|_\star \geq 1$ implies that $\mathfrak{D}(A)$ is a factor of some $\#\,\langle 7i+C_v(i)\rangle_v\,\#$, $1 \leq i \leq n$, or a factor of some $\langle j\rangle_\diamond$, $1 \leq j \leq 14n$.*

▶ **Lemma 5.** *There is a minimal grammar G for $w_\mathcal{G}$ such that, for every i, $1 \leq i \leq 14n$, there is a nonterminal with derivative $\langle i\rangle_\diamond$ and a nonterminal with derivative $\langle i\rangle_v$, and, for every i, $1 \leq i \leq n$, there is a nonterminal with derivative $\#\,\langle 7i+C_v(i)\rangle_v$ and a nonterminal with derivative $\langle 7i+C_v(i)\rangle_v\#$.*

Proof Sketch. Let G be a minimal grammar. Since $|u|_\star = 196n$, Lemma 4 implies that $|\beta| \geq 196n$, where β is the prefix of the compressed string producing u. Also by Lemma 4, every $\langle i\rangle_\diamond$ or $\langle i\rangle_v$ that is not a derivative of some rule, is produced by at least two nonterminals (we assume that there are k many such *bad* codewords). This implies that β contains at least $7\lceil\frac{k}{2}\rceil$ additional nonterminals (each codeword has 7 occurrences in u and the nonterminal not producing $\star$ can be used in the production of at most 2 bad codewords). Hence, $|\beta| \geq 196n + 7\lceil\frac{k}{2}\rceil$. For every bad codeword $x \in \{\langle i\rangle_\diamond, \langle i\rangle_v\}$, we can add a new rule $A_x \to \alpha_x$ with $|\alpha_x| = 3$ and $\mathfrak{D}(A_x) = x$ (this can be done by right sides of the form $d_j A d_j$, where A derives another codeword). This increases the size of the grammar by $3k$, but we can now produce every codeword of u by one nonterminal, which shortens β by $7\lceil\frac{k}{2}\rceil > 3k$.

We now add rules $\overleftarrow{V_i} \to \#V_{7i+C_v(i)}$, $\overrightarrow{V_i} \to V_{7i+C_v(i)}\#$, where $\mathfrak{D}(V_{7i+C_v(i)}) = \langle 7i+C_v(i)\rangle_v$, and use them, in addition to the rules for the codewords $\langle i\rangle_\diamond$, to obtain a new compressed string from $w_\mathcal{G}$ (where also every factor $\#\langle 7i+C_v(i)\rangle_v\#$ that has been produced before by a single nonterminal is compressed by a rule $\overleftrightarrow{V_i} \to \overleftarrow{V_i}\#$). Then, we erase all old rules with derivatives $\#f(7i+C_v(i))\star r_i$, $r_i \star f(7i+C_v(i))^R\#$. The deletion of these rules and the size of the new compressed string compensates the size increase of adding the new rules. ◀

▶ **Lemma 6.** *There is a minimal grammar G for $w_\mathcal{G}$ with the rules $\{r_{\diamond,i}, r_{v,i} \colon 1 \leq i \leq 14n\}$, where $r_{\diamond,i} = D_i \to d_i \star d_i$ and $r_{v,i} = V_i \to x_i \star x_i$, $1 \leq i \leq 7$, $r_{\diamond,i} = D_i \to g(i)[1]D_{h(i)}g(i)[1]$ and $r_{v,i} = V_i \to f(i)[1]V_{h(i)}f(i)[1]$, $8 \leq i \leq 14n$ with $h(i) = \frac{1}{7}(i - M(i,7))$, $\{\overleftarrow{V_i} \to \#V_{7i+C_v(i)}, \overrightarrow{V_i} \to V_{7i+C_v(i)}\# \colon 1 \leq i \leq n\}$, $\{\overleftrightarrow{V_i} \to \#\overrightarrow{V_i} \colon i \in \mathfrak{I}\}$, for an $\mathfrak{I} \subseteq \{1, \ldots, n\}$, and with the compressed string*

$\prod_{j=0}^{6}\left(\prod_{i=1}^{14n}(D_i\, V_{M(i+j,14n)})\right)\prod_{i=1}^{n}\left(\overleftarrow{V_i}\ \mathfrak{c}_1\ D_{7i-1}\right)\prod_{i=1}^{n}\left(\overleftarrow{V_i}\ \mathfrak{c}_2\ D_{7i-2}\right)\prod_{i=1}^{n}\left(\vec{V_i}\ D_{7i-2}\ \mathfrak{c}_1\right)$
$\prod_{i=1}^{n}\left(\vec{V_i}\ D_{7i-1}\ \mathfrak{c}_2\right)\prod_{i=1}^{n}(y_i\ D_{7i})\prod_{i=1}^{m-1}(z_i D_{7i+C_e(v_{j_{2i}},v_{j_{2i+1}})})z_m$,
where for every i, $1\le i\le n$, $y_i=\overleftrightarrow{V_i}$, *if* $i\in\mathfrak{I}$ *and* $y_i=\overleftarrow{V_i}\#$, *otherwise, and, for every* k, $1\le k\le m$, $z_k\in\{\overleftrightarrow{V}_{j_{2k-1}}\vec{V}_{j_{2k}},\overleftarrow{V}_{j_{2k-1}}\overleftrightarrow{V}_{j_{2k}}\}$ *if* $\{j_{2k-1},j_{2k}\}\cap\mathfrak{I}\neq\emptyset$, $z_k=\overleftarrow{V}_{j_{2k-1}}\overleftarrow{V}_{j_{2k}}\#$, *otherwise.*

Proof Sketch. Lemma 5 ensures nonterminals $V_i\to\alpha_i$ with $\mathfrak{D}(\alpha_i)=\langle i\rangle_v$. We now replace it by a rule $V_i\to x_i\star x_i$ or $V_i\to f(i)[1]V_{h(i)}f(i)[1]$, as described in the statement of the lemma. If $|\alpha_i|\ge 3$, this is fine, but if $|\alpha_i|=2$, we have to argue more carefully: we remove $V_i\to AB$ for which i is maximal and observe that this implies that either A or B cannot occur on any right side of a rule; thus, can be removed. Repeating this argument turns all rules with derivative $\langle i\rangle_v$ in the right form and a similar argument applies to the rules with derivatives $\langle i\rangle_\diamond$. Now it only remains to prove that the compressed string can be assumed to have the desired form. If we replace all $\langle i\rangle_v$, $\langle i\rangle_\diamond$, $\#\langle i\rangle_v$ and $\langle i\rangle_v\#$ in $w_{\mathcal{G}}$ by the respective nonterminals, then this produces a compressed string whose size may increase compared to the original one, but only by the number of factors $\#\langle i\rangle_v\#$ that have been compressed by a single nonterminal and are now compressed by $\overleftarrow{V_i}\#$. This can be repaired by simply adding a rule $\overleftrightarrow{V_i}\to\#\vec{V_i}$, resulting in the set $\mathfrak{I}$ mentioned in the statement of the lemma. ◀

Lemma 6 allows us to argue similarly as for the reduction from [4]: $\Gamma=\{v_i\colon i\in\mathfrak{I}\}$ is a vertex cover (if $\{v_i,v_j\}\in E$ is not covered, then adding $\overleftrightarrow{V_i}\to\#\vec{V_i}$ does not increase the size of the grammar) and $|G|=f(m,n)+|\Gamma|$, where f is a polynomial. Furthermore, a vertex cover Γ translates into a grammar of size $f(m,n)+|\Gamma|$, by setting $\mathfrak{I}=\{i\colon v_i\in\Gamma\}$. Thus, $\mathcal{G}$ has a vertex cover Γ iff there is a grammar G for $w_{\mathcal{G}}$ with $|G|\le f(m,n)+|\Gamma|$.

▶ **Theorem 7.** SGP *is* NP*-complete, even for alphabets of size* 24.

4 Minimal Grammars with a Bounded Number of Nonterminals

A natural follow-up question to the hardness for fixed alphabets is whether polynomial-time solvability is possible if instead the cardinality of the nonterminal alphabet N is bounded. In this section, we answer this question in the affirmative by representing words $w\in\Sigma^*$ as graphs $\Phi_{\mathrm{m}}(w)$ and $\Phi_1(w)$, such that smallest independent dominating sets of these graphs correspond to a smallest grammar and a smallest 1-level grammar, respectively, for w.

We first define $\Phi_1(w)$ and then derive $\Phi_{\mathrm{m}}(w)$ from $\Phi_1(w)$. Let $\Phi_1(w)=(V,E)$ be defined by $V=V_1\cup V_2\cup V_3$ and $E=E_1\cup E_2\cup E_3$, where

$$V_1=\{(i,j)\colon 1\le i\le j\le|w|\},\qquad E_1=\{\{(i_1,j_1),(i_2,j_2)\}\colon i_1\le i_2\le j_1\},$$
$$V_2=\mathsf{F}_{\ge 2}(w),\qquad E_2=\{\{w[i..j],(i,j)\}\colon 1\le i<j\le|w|\},$$
$$V_3=\{(u,i)\colon u\in V_2,0\le i\le|u|\},\qquad E_3=\{\{u,(u,i)\}\colon u\in V_2,0\le i\le|u|\}.$$

Intuitively speaking, the vertices of V_1 represent every factor by its start and end position, whereas V_2 contains exactly one vertex per factor of length at least 2. Every $u\in V_2$ is connected to (i,j), if and only if $w[i..j]=u$. Vertices (i,j), (i',j') are connected if they refer to overlapping factors. For every $u\in V_2$, there are $|u|+1$ special vertices in V_3 that are only connected with u. Consequently, $\Phi_1(w)$ consists of $|w|$ layers, where the i^{th} layer contains the vertices $(j,j+(i-1))\in V_1$, $1\le j\le|w|-(i-1)$, the vertices $\{u\in V_2\colon|u|=i\}$ and the vertices $\{(u,j)\in V_3\colon|u|=i,0\le j\le|u|\}$ (see Figure 1 for an illustration).

▶ **Lemma 8.** *Let* $w\in\Sigma^*$, $k\ge 1$. *There is an independent dominating set* D *of cardinality* k *for* $\Phi_1(w)$ *if and only if there is a* 1*-level grammar* G *for* w *with* $|G|=k-|\,\mathsf{F}_{\ge 2}(w)|$.

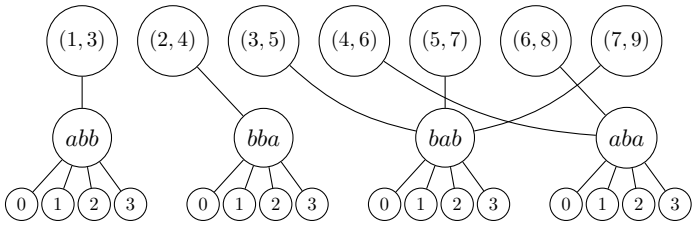

Figure 1 The third layer of $\Phi_1(abbababab)$ (edges E_1 omitted).

Proof Sketch. For an independent dominating set D of $\Phi_1(w)$, $V_1 \cap D$ induces a factorisation of w. For every $(i,j) \in D$, $w[i..j] \notin D$, which implies that all $|w[i..j]|+1$ many V_3-neighbours of $w[i..j]$ are in D. Now a 1-level grammar can be obtained by constructing rules for all $V_2 \setminus D$. Analogously, a 1-level grammar translates into an independent dominating set. ◀

In order to extend this idea to the multi-level case, what comes to mind is to somehow represent the vertices $u \in V_2$ again by graph structures of the type $\Phi_1(u)$ and repeating this step, which considerably increases the size of the graph. Fortunately, it turns out that a surprisingly simple modification of $\Phi_1(w)$ is sufficient. For a word $w \in \Sigma^*$, let $\Phi_{\mathrm{m}}(w) = (V, E)$ be defined as follows. Let $V = V_1 \cup V_2 \cup V_3 \cup V_4$, where V_1 and V_2 are defined as for $\Phi_1(w)$, $V_3 = \{(u,0) \colon u \in V_2\}$ and $V_4 = \bigcup_{u \in V_2} V_{4,u}$ with $V_{4,u} = \{(u,i,j) \colon 1 \le i \le j \le |u|, u[i..j] \neq u\}$ for every $u \in V_2$. Moreover, $E = E_1 \cup E_2 \cup E_3 \cup E_4 \cup E_5$, where E_1 and E_2 are defined as for $\Phi_1(w)$, $E_3 = \{\{u,(u,0)\} \colon u \in V_2\} \cup \{\{u,(u,i,j)\} \colon u \in V_2, (u,i,j) \in V_{4,u}\}$, $E_4 = \bigcup_{u \in V_2} E_{4,u}$, where, for every $u \in V_2$, $E_{4,u} = \{\{(u,i_1,j_1),(u,i_2,j_2)\} \subseteq V_{4,u} \colon i_1 \le i_2 \le j_1\}$ and $E_5 = \{\{u,(v,i,j)\} \colon u,v \in V_2, v[i..j] = u, u \neq v\}$.

Intuitively speaking, $\Phi_{\mathrm{m}}(w)$ differs from $\Phi_1(w)$ in the following way. We add to every vertex $u \in V_2$ a subgraph $(V_{4,u}, E_{4,u})$, which is completely connected to u and which represents u in the same way as the subgraph (V_1, E_1) of $\Phi_1(w)$ represents w, i. e., factors $u[i..j]$ are represented by (u,i,j) and edges represent overlappings. Moreover, if a $u \in V_2$ is a factor of some $v \in V_2$, then there is an edge from u to all the vertices $(v,i,j) \in V_{4,v}$ that satisfy $v[i..j] = u$. Finally, every $u \in V_2$ is also connected with an otherwise isolated vertex $(u,0) \in V_3$. See Figure 2 for a partial illustration of a $\Phi_{\mathrm{m}}(w)$.

▶ **Lemma 9.** *Let $w \in \Sigma^*$, $k \ge 1$. There is an independent dominating set D of cardinality k for $\Phi_{\mathrm{m}}(w)$ if and only if there is a grammar G for w with $|G| = k - |\,\mathsf{F}_{\ge 2}(w)|$.*

Proof Sketch. The correspondence of independent dominating sets D for $\Phi_{\mathrm{m}}(w)$ and grammars for w is similar as in the 1-level case. Again, $D \cap V_1$ induces a factorisation of w, and, in the same way, for every $u \in V_2 \setminus D$ (i. e., the factors for which rules will be constructed), $D \cap V_{4,u}$ induces a factorisation of u. Each $(v,i,j) \in D \cap V_{4,u}$ with $|v| \ge 2$ is connected to $v \in V_2$, which implies that $v \notin D$; thus, v will also be represented by a rule and so on. ◀

The proofs of Lemmas 8 and 9 also show how an independent dominating set D of $\Phi(w) \in \{\Phi_1(w), \Phi_{\mathrm{m}}(w)\}$ translates into a grammar for w, which, in the following, we will denote by $\mathsf{G}(D)$. Consequently, we can solve the smallest grammar problem by computing minimal independent dominating sets. Unfortunately, this is a hard problem, even for quite restricted graph classes [15, Theorem 13]. However, $\Phi(w)$ may have structural features that could be exploited in this regard, e. g., it is a 2-interval graph (see [8]).

Our algorithmic application is based on the following observation. If we are looking for a grammar $G = (N, \Sigma, R, cs)$ with $\{\mathfrak{D}(A) \colon A \in N\} = F$, for some set $F \subseteq \mathsf{F}_{\ge 2}(w)$, then we need an independent dominating set D with $(\mathsf{F}_{\ge 2}(w) \setminus F) \subseteq D$ and $F \cap D = \emptyset$. Obviously,

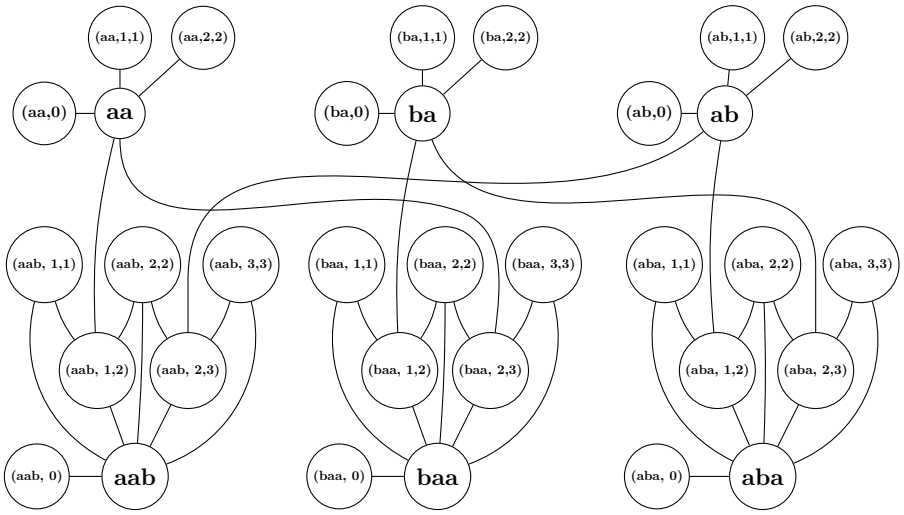

Figure 2 Second and third layer of $\Phi_{\mathrm{m}}(abaabaa)$ (vertices V_1 and edges $E_1 \cup E_2$ omitted).

D is the disjoint union of $(\mathsf{F}_{\geq 2}(w) \setminus F)$ and an independent dominating set D' for the graph $\mathcal{H} = \Phi(w) \setminus (N[\mathsf{F}_{\geq 2}(w) \setminus F] \cup F)$,[7] which is necessarily an interval graph; thus, a smallest independent dominating set for it can be efficiently computed (see [6]). For a word w and a set $F \subseteq \mathsf{F}_{\geq 2}(w)$, we define $\mathsf{MinIDS}(w, F) = D_{\mathcal{H}} \cup (\mathsf{F}_{\geq 2}(w) \setminus F)$, where $D_{\mathcal{H}}$ is a smallest independent dominating set for $\mathcal{H} = \Phi_{\mathrm{m}}(w) \setminus (N[\mathsf{F}_{\geq 2}(w) \setminus F] \cup F)$, and $\mathsf{1L\text{-}MinIDS}(w, F)$ is defined analogously by using $\Phi_1(w)$ instead of $\Phi_{\mathrm{m}}(w)$. In this way, for any set $F \subseteq \mathsf{F}_{\geq 2}$, we can compute a grammar that is minimal among all grammars that have rules for exactly the factors in F (this is also an interesting problem in its own right, e. g., if a compressed word is extended by a new part that should be compressed by already existing rules).

▶ **Lemma 10.** *Let $w \in \Sigma^+$ and $F \subseteq \mathsf{F}_{\geq 2}(w)$. $\mathsf{MinIDS}(w, F)$ and $\mathsf{1L\text{-}MinIDS}(w, F)$ can be computed in time $\mathcal{O}(|w|^6)$ and $\mathcal{O}(|w|^4)$, respectively. Furthermore, $\mathsf{G}(\mathsf{MinIDS}(w, F))$ is a minimal grammar for w and $\mathsf{G}(\mathsf{1L\text{-}MinIDS}(w, F))$ is a minimal 1-level grammar for w.*

If instead of a set F of factors, we are only given an upper bound k on $|N|$, then we can compute a minimal grammar by enumerating all $F \subseteq \mathsf{F}_{\geq 2}(w)$ with $|F| \leq k$ and computing $\mathsf{G}(\mathsf{MinIDS}(w, F))$. This shows that minimal grammars can be computed in polynomial time if the number of nonterminals is bounded.

▶ **Theorem 11.** *Let $w \in \Sigma^*$ and $k \in \mathbb{N}$. A grammar (1-level grammar, resp.) for w with at most k rules that is minimal among all grammars (1-level grammars, resp.) for w with at most k rules can be computed in time $\mathcal{O}(|w|^{2k+6})$ ($\mathcal{O}(|w|^{2k+4})$, resp.).*

The next question is whether these problems are also fixed-parameter tractable with respect to the number of nonterminals.[8] Unfortunately, this seems unlikely, since, as stated by the next result, these parameterisations of 1-SGP and SGP are $\mathsf{W}[1]$-hard.

▶ **Theorem 12.** 1-SGP *and* SGP *parameterised by $|N|$ are $\mathsf{W}[1]$-hard.*

[7] $N[v]$ is the closed neighbourhood of vertex v and for $C \subseteq V$, $N[C] = \bigcup_{v \in C} N[v]$.

[8] For unexplained concepts of parameterised complexity, we refer to Downey and Fellows [5].

This can be proven by reducing from independent set, i. e., a graph $\mathcal{G} = (\{v_1, v_2, \ldots, v_n\}, E)$ and $k \in \mathbb{N}$ is transformed into the word $w = \prod_{\{v_i,v_j\}\in E}(\#v_i\#v_j\#\diamond) \prod_{i=1}^{n}(\#v_i\#\diamond)^{n-|N(v_i)|}$, where $N(v_i)$ is the neighbourhood of v_i and every occurrence of $\diamond$ stands for a distinct symbol. To see the the correctness of this reduction, it is sufficient to observe that the vertices v_i of an independent set for G correspond to the rules of a grammar of form $A_i \to \#v_i\#$.

5 Exact Exponential Time Algorithms

Computing $\mathsf{G}(\mathsf{MinIDS}(w, F))$, for all $F \subseteq \mathsf{F}_{\geq 2}(w)$, yields a simple brute-force algorithm with a running time in $\mathcal{O}(2^{|w|^2})$. Another obvious approach is to enumerate all ordered trees with $|w|$ leaves (for each such tree T, an optimal grammar whose derivation tree has the structure T can be easily computed), which can be done in time $\mathcal{O}(8^{|w|})$.[9] In the following, we shall give more sophisticated exact exponential-time algorithms with running times in $\mathcal{O}^*(1.8392^{|w|})$, for the 1-level case, and $\mathcal{O}^*(3^{|w|})$, for the multi-level case.

Let $G = (N, \Sigma, R, cs)$ be a grammar for w and let $\alpha = A_1 \ldots A_k$, $A_i \in (\Sigma \cup N)$, $1 \leq i \leq k$. The *factorisation of* $\mathfrak{D}(\alpha)$ *induced by* α is the tuple $(\mathfrak{D}_G(A_1), \ldots, \mathfrak{D}_G(A_k))$. Furthermore, the factorisation of w induced by cs is called the *factorisation of* w *induced by* G.

5.1 The 1-Level Case

Let $q = (u_1, u_2, \ldots, u_k)$ be a factorisation for a word w and let $\Gamma_q = \{u_i \colon 1 \leq i \leq k, |u_i| \geq 2\}$ and let the 1-level grammar $G_q = (N_q, \Sigma, R_q, cs_q)$ be defined by $R_q = \{(A_u, u) \colon u \in \Gamma_q\}$, $N_q = \{A_u \colon u \in \Gamma_q\}$ and $cs_q = B_1 \ldots B_k$ with $B_j = A_{u_j}$, if $u_j \in \Gamma_q$ and $B_j = u_j$, otherwise.

▶ **Lemma 13.** *For any factorisation $q = (u_1, u_2, \ldots, u_k)$ for w, G_q is minimal among all 1-level grammars for w that induce the factorisation q.*

Choosing the smallest among all grammars $\{G_q \colon q \text{ is a factorisation of } w\}$ yields an $\mathcal{O}^*(2^n)$ algorithm for 1-SGP. However, it is not necessary to enumerate factorisations that contain at least two consecutive factors of length 1, which improves this result as follows.

▶ **Theorem 14.** 1-SGP *can be solved exactly in polynomial space and in time $\mathcal{O}^*(1.8392^{|w|})$.*

5.2 The Multi-Level Case

The obvious idea for a dynamic programming algorithm is to extend a smallest i-level grammar by a new level in order to obtain a smallest $(i+1)$-level grammar. However, this approach does not seem to work if we take the levels of a grammar to be $cs, \mathsf{D}(cs), \mathsf{D}(\mathsf{D}(cs)), \ldots, w$ (note that these are also the levels of the derivation tree). Intuitively speaking, the problem is that if we try to either add a new level on top (i. e., a new compressed string) of the grammar or at the bottom (by further compressing the terminal right sides of the last rules applied), then this decision is not local, since it is possible that rules to be added are already used somewhere else in the grammar. So we need to define levels in such a way that all occurrences of a nonterminal are on the same level.

For a d-level grammar $G = (N, \Sigma, R, cs)$, let $N_1, \ldots, N_d$ be the partition of N into $N_i = \{A \in N \colon (\mathsf{D}_G^i(A) \in \Sigma^+) \wedge (\mathsf{D}_G^{i-1}(A) \notin \Sigma^+)\}$ and let $\mathsf{L}_i \colon (N \cup \Sigma)^* \to (N \cup \Sigma)^*$, $1 \leq i \leq d$, be component-wise defined by $\mathsf{L}_i(x) = \mathsf{D}(x)$, if $x \in N_i$ and $\mathsf{L}_i(x) = x$, otherwise.

[9] There are $C_{|w|-1} \leq 4^{|w|-1}$ ordered binary trees with $|w|$ leaves, where $C_{|w|-1}$ is the $(|w|-1)^{\text{th}}$ Catalan number, and every ordered tree can be obtained from a binary one by contracting some of its edges.

Taking the strings $(\mathsf{L}_{i+1} \circ \mathsf{L}_{i+2} \circ \cdots \circ \mathsf{L}_d)(cs)$, which contain all occurrences of nonterminals N_i, as the levels of the grammar, we are able to define a dynamic programming algorithm.[10]

▶ **Theorem 15.** SGP *can be solved exactly in time and space* $\mathcal{O}^*(3^{|w|})$.

Proof Sketch. With the help of the mappings L_i, we can define the term *profit* for rules from a d-level grammar $G = (N, \Sigma, R, cs)$ as follows. The profit for a rule $A \to \alpha$ with $A \in N_d$ can be defined like in the 1-level case, i. e., $\mathsf{p}(A) = |cs|_A(|\alpha| - 1) - |\alpha|$, considering that removing this rule and replacing each occurrence of A in cs by α increases the size of the grammar by $|cs|_A(|\alpha| - 1) - |\alpha|$. Inductive use of this argument allows us to define the profit of any rule $A \to \alpha$ with $A \in N_i$ by $\mathsf{p}(A) := |(\mathsf{L}_{i+1} \circ \mathsf{L}_{i+2} \circ \cdots \circ \mathsf{L}_d)(cs)|_A(|\alpha| - 1) - |\alpha|$. This allows us to compute the size of a G by $|w| - \sum_{A \in N} \mathsf{p}(A)$. The dynamic programming algorithm runs trough steps $i = 1, 2, \ldots, \frac{w}{2}$ and in step i, it considers all possibilities for two factorisations q_{i-1} and q_i of w induced by $(\mathsf{L}_i \circ \mathsf{L}_{i+1} \circ \cdots \circ \mathsf{L}_d(cs)$ and $(\mathsf{L}_{i+1} \circ \cdots \circ \mathsf{L}_d)(cs)$, respectively (note that this implies $q_{i-1} \preceq q_i$). The differences between q_{i-1} and q_i implicitly define N_i. Let $q_i = (v_1, v_2, \ldots, v_k)$ and let $q_{i-1} = (u_1, u_2, \ldots, u_\ell)$, i. e., for some j_i, $0 \leq i \leq k$, with $1 = j_0 < j_1 < \ldots < j_k = \ell + 1$, $(u_{j_{i-1}}, u_{j_{i-1}+1}, \ldots, u_{j_i-1})$ is a factorisation of v_i, $1 \leq i \leq k$. If $j_s - j_{s-1} > 1$ for some $1 \leq s \leq k$, N_i contains a nonterminal A with $|\,\mathsf{D}(A)| = j_s - j_{s-1}$ and $\mathfrak{D}(A) = v_s$. The term $|\,\mathsf{L}_i \circ \mathsf{L}_{i+1} \circ \cdots \circ \mathsf{L}_d)(cs)|_A$ is also implicitly given by counting how often the sequence of factors $(u_{j_{s-1}+1}, \ldots, u_{j_s})$ independently occurs in q_{i-1} and is combined into one single factor in q_i, i.e.: $|\{t\colon (u_{j_{t-1}+1}, \ldots, u_{j_t}) = (u_{j_{s-1}+1}, \ldots, u_{j_s})\}|$. This allows to calculate the profit of the rule for A without knowing the exact structure of the rules for nonterminals in N_j with $j \neq i$. By Lemma 13, this choice of nonterminals for N_i is optimal for the fixed induced factorisations, which means that a search among all choices for q_{i-1} and q_i yields a minimal i-level grammar for w. The running time of this algorithm is dominated by enumerating all pairs q_{i-1} and q_i of factorisations of w. However, due to $q_{i-1} \preceq q_i$, these pairs can be compressed as vectors $\{0, 1, 2\}^{|w|-1}$ (the entries denote whether the corresponding position in w is factorised by both, only one or none of the factorisations). Hence, enumerating these pairs of vectors can be done in time $\mathcal{O}(3^{|w|})$. ◀

6 Conclusions

We conclude this work by deriving some parameterised complexity results.[11] The shortest-grammar problem (1-level and multi-level) is Para-NP-hard with respect to $|\Sigma|$, it is in XP with respect to $|N|$, but also W[1]-hard, so most likely not in FPT. Furthermore, the hardness of 1-SGP shows that bounding or parameterising by the number of levels does not help either. However, if we parameterise by both $|\Sigma|$ and $\ell = \max\{|\,\mathfrak{D}(A)|\colon A \in N\}$, then it is sufficient to compute $\mathsf{G}(\mathsf{MinIDS}(w, F))$ for every set $F \subseteq \{u\colon u \in \Sigma^+, |u| \leq \ell\}$, which, since the number of such sets is bounded by the parameters, yields an fpt-algorithm. A probably more interesting combination of parameters, for which the existence of an fpt-algorithm is still open, would be $|\Sigma|$ and $|N|$.

The most interesting question (also from a practical point of view) left open is whether it is possible to compute minimal grammars for small (especially binary) alphabets in polynomial-time. The substantial effort that was necessary to prove Theorem 7 suggests that answering this question in the negative might be difficult. On the other hand, it is not apparent how a small alphabet could help in order to efficiently compute smallest grammars and, if

[10] The composition $(f \circ g)$ of mapping $f : A \to A$, $g : A \to A$ is defined by $(f \circ g)(a) = f(g(a))$.
[11] For unexplained concepts of parameterised complexity, we refer to Downey and Fellows [5].

this is possible, it seems that deeper combinatorial insights with respect to grammar-based compression are necessary.

References

1 T. Akutsu. A bisection algorithm for grammar-based compression of ordered trees. *Information Processing Letters*, 110(18-19):815–820, 2010.

2 J. Arpe and R. Reischuk. On the complexity of optimal grammar-based compression. In *2006 Data Compression Conference (DCC)*, pages 173–182. IEEE Computer Society, 2006.

3 P. Berman, M. Karpinski, L. L. Larmore, W. Plandowski, and W. Rytter. On the complexity of pattern matching for highly compressed two-dimensional texts. *Journal of Computer and System Sciences*, 65(2):332–350, 2002.

4 M. Charikar, E. Lehman, D. Liu, R. Panigrahy, M. Prabhakaran, A. Sahai, and A. Shelat. The smallest grammar problem. *IEEE Transactions on Information Theory*, 51(7):2554–2576, 2005.

5 R. G. Downey and M. R. Fellows. *Fundamentals of Parameterized Complexity*. Texts in Computer Science. Springer, 2013.

6 M. Farber. Independent domination in chordal graphs. *Operations Research Letters*, 1(4):134–138, 1982.

7 M. R. Garey, D. S. Johnson, and L. Stockmeyer. Some simplified NP-complete graph problems. *Theoretical Computer Science*, 1:237–267, 1976.

8 M. Jiang and Y. Zhang. Parameterized complexity in multiple-interval graphs: Domination, partition, separation, irredundancy. *Theoretical Computer Science*, 461:27–44, 2012.

9 J. C. Kieffer and E.-H. Yang. Grammar-based codes: A new class of universal lossless source codes. *IEEE Transactions on Information Theory*, 46(3):737–754, 2000.

10 J. C. Kieffer, E.-H. Yang, G. J. Nelson, and P. C. Cosman. Universal lossless compression via multilevel pattern matching. *IEEE Transactions on Information Theory*, 46(4):1227–1245, 2000.

11 M. Lohrey. Algorithmics on SLP-compressed strings: A survey. *Groups, Complexity, Cryptology*, 4:241–299, 2012.

12 M. Lohrey and S. Maneth. The complexity of tree automata and XPath on grammar-compressed trees. *Theoretical Computer Science*, 363(2):196–210, 2006.

13 M. Lohrey, S. Maneth, and R. Mennicke. XML tree structure compression using RePair. *Information Systems*, 38(8):1150–1167, 2013.

14 M. Lohrey, S. Maneth, and M. Schmidt-Schauß. Parameter reduction and automata evaluation for grammar-compressed trees. *Journal of Computer and System Sciences*, 78(5):1651–1669, 2012.

15 D. F. Manlove. On the algorithmic complexity of twelve covering and independence parameters of graphs. *Discrete Applied Mathematics*, 91:155–175, 1999.

16 C. G. Nevill-Manning. *Inferring Sequential Structure*. PhD thesis, University of Waikato, NZ, 1996.

17 C. G. Nevill-Manning and I. H. Witten. Identifying hierarchical structure in sequences: A linear-time algorithm. *Journal of Artificial Intelligence Research*, 7:67–82, 1997.

18 W. Rytter. Application of Lempel-Ziv factorization to the approximation of grammar-based compression. *Theoretical Computer Science*, 302:211–222, 2003.

19 C. E. Shannon. A theorem on coloring the lines of a network. *J. Math. Physics*, 28:148–151, 1949.

20 S. Skulrattanakulchai. Δ-list vertex coloring in linear time. *Information Processing Letters*, 98(3):101–106, 2006.

21 J. A. Storer. NP-completeness results concerning data compression. Technical Report 234, Dept. Electrical Engineering and Computer Science, Princeton University, USA, November 1977.

22 J. A. Storer and T. G. Szymanski. Data compression via textural substitution. *Journal of the ACM*, 29(4):928–951, 1982.

23 E.-H. Yang and J. 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.