On the Complexity of Grammar-Based Compression over Fixed Alphabets

Katrin Casel\(^1\), Henning Fernau\(^1\), Serge Gaspers\(^2,3\), Benjamin Gras\(^4\), and Markus L. Schmid\(^1\)

\(^1\) Trier University, Fachbereich IV – Abteilung Informatikwissenschaften, D-54286 Trier, Germany, \{Casel, Fernau, MSchmid\}@uni-trier.de
\(^2\) UNSW Australia, Sydney, Australia
sergeg@cse.unsw.edu.au
\(^3\) Data61 (formerly: NICTA), CSIRO, Sydney, Australia
\(^4\) École Normale Superieure de Lyon – Département Informatique
benjamin.gras@ens-lyon.fr

Abstract

It is shown that the shortest-grammar problem remains \(\text{NP}\)-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 \(O(3^n)\) 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.xxx.yyy.p

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 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 \([16, 17]\), and Kieffer et al. \([9, 10, 23]\) 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 \([21, 22]\) is also equivalent to grammar-based compression.

The success of grammars with respect to algorithms on compressed strings is due to the fact that they cover many compression schemes from practice (most notably, the family

---

\(^1\) Such context-free grammars are also called *straight-line programs* in the literature.
Complexity of Grammar-Based Compression

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 [1,12–14]) 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 \( \text{NP} \)-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 \( \mathcal{O}(\log (|w|/m^*)) \) (see [4,18]), 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 \( \text{P} \neq \text{NP} \) (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 \( \text{NP} \)-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 \( \text{FPT} \)-algorithm with respect to this 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.

---

\(^2\) Note that this negative result also transfers to the shortest-grammar problem for trees.
2 Preliminaries

Let \( \mathbb{N} = \{1, 2, 3, \ldots\} \). By \( |A| \), we denote the cardinality of a set \( A \). Let \( \Sigma \) be a finite alphabet of symbols. A word or string (over \( \Sigma \)) is a sequence of symbols from \( \Sigma \). For any word \( w \) over \( \Sigma \), \( |w| \) denotes the length of \( w \) and \( \varepsilon \) denotes the empty word, i.e., \( |\varepsilon| = 0 \). The symbol \( \Sigma^+ \) denotes the set of all non-empty words over \( \Sigma \) 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, \( F(w) = \{u : u \text{ is a factor of } w\} \) and \( F_{\geq 2}(w) = \{u : u \in F(w), |u| \geq 2\} \). 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 \((u_{j_1-1}, u_{j_1-1+2}, \ldots, u_{j_1})\) is a factorisation of \( v_j \), \( 1 \leq i \leq m \) for some \( j_1 \) such that \( 0 = j_0 < j_1 < \ldots < j_m = k \).

**Grammars:** A context-free grammar is a tuple \( G = (N, \Sigma, R, S) \), where \( N \) is the set of nonterminals, \( \Sigma \) 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 \( A \rightarrow w \)). A context-free grammar \( G = (N, \Sigma, R, S) \) is a singleton grammar if \( R \) is a total function \( N \rightarrow (N \cup \Sigma)^+ \) and the relation \(((A, B) : (A, w) \in R, |a|_B \geq 1)\) is acyclic.

For a singleton grammar \( G = (N, \Sigma, R, S) \), let \( D_G : (N \cup \Sigma) \rightarrow (N \cup \Sigma)^+ \) be defined by \( D_G(A) = R(A), A \in N, \) and \( D_G(a) = a, a \in \Sigma \). We extend \( D_G \) to a morphism \( (N \cup \Sigma)^+ \rightarrow (N \cup \Sigma)^+ \) by setting \( D_G(a_1a_2 \ldots a_n) = D_G(a_1)D_G(a_2) \ldots D_G(a_n) \), for \( a_i \in (N \cup \Sigma), 1 \leq i \leq n \). Furthermore, for every \( \alpha \in (N \cup \Sigma)^+ \), we set \( D_G^{\alpha}(a) = D_G(a) \), \( D_G^{\alpha}(a) = D_G(D_G^{k-1}(a)), \) for every \( k \geq 2 \), and \( D_G(\alpha) = \lim_{k \to \infty} D_G^{k}(\alpha) \) is the derivative of \( \alpha \). By definition, \( D_G(\alpha) \) exists for every \( \alpha \in (N \cup \Sigma)^+ \) and is an element from \( \Sigma^+ \). The size of the singleton grammar \( G \) is defined by \(|G| = \sum_{A \in N} |D_G(A)| \) and its number of levels is \( \min\{k : D_G^k(S) = D_G(S)\} \). 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 \( \mathcal{D}(G) = \mathcal{D}(S) \) and say that \( G \) is a grammar for \( \mathcal{D}(G) \). In the tuple \((N, \Sigma, R, S)\), we sometimes replace \( S \) directly by \( \mathcal{D}(S) \), 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 \( p(A) = |cs|_A(|\alpha| - 1) - |\alpha| \). Intuitively speaking, if all occurrences of \( A \) in \( cs \) are replaced by \( \alpha \) and the rule \( A \rightarrow \alpha \) is deleted, then the size of the grammar increases by exactly \( p(A) \). Consequently, \(|G| = |\mathcal{D}(G)| - \sum_{A \in N} p(A) \).

A grammar \( G \) is minimal if \(|G| = \min\{|G'| : G' \text{ is a grammar for } \mathcal{D}(G)\} \) 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 \( \mathcal{D}(G) = w \) 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 \( A_1A_2 \ldots A_{n-1}A_{n-1}0 \), 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 \( 1(A_1)^2(A_2)^2 \ldots \). 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 \( O(\sqrt{|w|}) \), but, generally, grammars are capable of exponential compression rates (see [4]). Aiming for such exponential compression, it seems worthwhile to represent every unary factor \( 0^2 \ell \), \( 1 \leq \ell \leq k \), by a nonterminal \( B_\ell \) (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 \( B_3B_2B_0 \)). 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 \( 0^{2\ell} \) by nonterminals \( B_\ell \). More precisely, for every \( \ell \), \( 1 \leq \ell \leq k-2 \), we represent \( 0^{2\ell} \) by an individual nonterminal \( B_\ell \) and, in addition, we use rules \( A_1 \to 010 \), \( A_i \to 0A_{i-1}0 \), \( 2 \leq i \leq \frac{n}{2} \). 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 \( 1(A_1)^2 \ldots (A_2)^2B_{k-2}(A_1B_{k-1})^2 \ldots (A_{\frac{n}{2}-1}B_{k-1})^2A_{\frac{n}{2}}(B_{k-2})^3 \). The size of this grammar \( G_4 \) is only \( \frac{9n}{2} + 2k - 2 \).

### 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 3 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
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.

Proof Sketch. A smallest grammar for $w$ produces the two parts to the left and right of $\star$ independently, since $|w|_a = 1$. According to Lemma 2, the right side $[\#\sigma^n]$ is best compressed by $n^3$ occurrences of a nonterminal $D$ with derivative $[\sigma]$ 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 $[\sigma]$ by $D$. Having established this basic property, it is then possible to show that the remaining rules have derivative $\#\sigma^i$, $\sigma^i\#$ or $\#\sigma^i\#$. The grammar is smallest, if every edge $\#u_{j_{2i-1}}u_{j_{2i}}\#$ 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 The bound of Lemma 2 is tight, e. g., consider $a^{n^2}$ and a grammar with rules $S \to A^n$ and $A \to a^n$. 

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

$$w = \prod_{i=1}^{n} (\#\sigma_i\#[\sigma])^{[\log(n)]} + 3 \prod_{i=1}^{n} (\#\sigma_i\#[\sigma])^{[\log(n)]} + 1 \prod_{i=1}^{n} (\#\sigma_{2i-1}\#\sigma_{2i}\#[\sigma])^2 \* [\sigma]^{n^3}.$$ 

Theorem 3. 1-SGP is NP-hard, even for $|\Sigma| = 5$.
In order to formally define the reduction, we first give some preparatory definitions. Let \( \Sigma = \{x_1, \ldots, x_7, d_1, \ldots, d_7, \#_1, \#_2, \#_3, \#_4, \#_5, \#_6, \#_7\} \) be an alphabet of size 24. The function \( M : \mathbb{N} \times \mathbb{N} \to \mathbb{N} \) is defined by \( M(q, k) \): \( q \geq k \), if \( q \% k \neq 0 \) and \( M(q, k) = k \), otherwise. Let the functions \( f : \mathbb{N} \to \{x_1, \ldots, x_7\}^+ \) and \( g : \mathbb{N} \to \{d_1, \ldots, d_7\}^+ \) be defined by \( f(q) := x_{a_1}x_{a_2} \ldots x_{a_k} \) 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 \( q = \sum_{i=0}^{k} a_i^7 \) is satisfied.\(^5\) For every \( i \in \mathbb{N} \), let \( (i)_v := f(i) \ast f(i)^R \) and \( (i)_o := g(i) \ast g(i)^R \). The factors \( (i)_v \) and \( (i)_o \) are called codewords; \( (i)_v \) represents a vertex \( v \), while \( (i)_o \) 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 \( (n)_v \) and \( (n')_v \) do not share any prefixes or suffixes (and the same holds for the words \( (n)_o \)).

Let \( G = (V, E) \) be a subcubic graph (i.e., a graph with maximum degree 3) with \( V = \{v_1, \ldots, v_n\} \) and \( E = \{(v_{j-1}, v_{j}) : 1 \leq i \leq m\} \) (note that the vertex cover problem remains \( \text{NP} \)-hard if restricted to subcubic graphs (see \([7]\))). Let \( G' = (V, E') \) be the multi-graph defined by \( E' = \{(v_{j-1}, v_{j}), 1 \leq i \leq m - 1\} \). 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 \( \lceil \frac{3}{4}m \rceil \) colours, where \( m \) is the maximum degree of the multi-graph. Since \( G \) is subcubic, the maximum degree of \( G' \) is three and we can compute a proper edge-colouring \( C'_v : E' \to \{1, 2, 3, 4\} \) for \( G' \) with colours \( \{1, 2, 3, 4\} \). Let \( G^2 = (V, E'' \in \{\{u, v\} \in V \setminus \{u, v\}, u \neq v\} \). Since \( G \) is subcubic, \( G^2 \) has maximum degree at most six. Let \( C_v : \{1, \ldots, n\} \to \{1, 2, 3, 4, 5, 6, 7\} \) be a proper vertex-colouring (defined over the vertex-indices of \( V = \{v_1, \ldots, v_n\} \)) for \( G^2 \) 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 \( u_w = uwv \) be the word representing \( G \), where \( u, v, w \in \Sigma^+ \) are defined as follows.\(^6\)

\[
\begin{align*}
u &= \prod_{j=0}^{m-1} \left( \prod_{i=1}^{14n} (i)_o \langle M(i + j, 14n) \rangle_v \right) \#_1 \\
v &= \prod_{i=1}^{m-1} (\# \langle \#_7 + C_v(i) \rangle_v e_1 \langle \#_7 - 1 \rangle_o) \#_2 \prod_{i=1}^{m-1} (\# \langle \#_7 + C_v(i) \rangle_v e_2 \langle \#_7 - 2 \rangle_o) \#_3 \\
&\quad \prod_{i=1}^{m-1} (\# \langle \#_7 + C_v(i) \rangle_v \# \langle \#_7 - 1 \rangle_o) \#_4 \prod_{i=1}^{m-1} (\# \langle \#_7 + C_v(i) \rangle_v \# \langle \#_7 - 1 \rangle_o) \#_5 \\
&\quad \prod_{i=1}^{m-1} (\# \langle \#_7 + C_v(i) \rangle_v \# \langle \#_7 \rangle_o) \#_6 \\
w &= \prod_{i=1}^{m-1} (\# \langle \#_7j_{2i-1} + C_v(j_{2i-1}) \rangle_v \# \langle \#_7j_{2i} + C_v(j_{2i}) \rangle_v \# \langle \#_7i + C_v(v_{j_{2i}}, v_{j_{2i+1}}) \rangle_o) \\
&\quad \# \langle \#_7j_{2i-1} + C_v(j_{2i-1}) \rangle_v \# \langle \#_7j_{2i} + C_v(j_{2i}) \rangle_v \#\end{align*}
\]

The next lemma states that any factor of \( w_G \) is not repeated, if it spans over the \( \ast \) of some codeword \((i)_v \) or \((i)_o \) and also reaches over the boundaries of this codeword into some other factor. This property, which can be proven in a straightforward, but rather cumbersome analysis, is crucial for the correctness of the reduction and also responsible for the complicated structure of \( w_G \). 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 \( \#_1, e_1, e_2, \#_3, \ldots, \#_6 \) by the fact that codewords \((i)_v \) and \((i)_o \) start and end with \( x_{M(i, 7)} \) and \( d_{M(i, 7)} \), respectively, and, for the part \( w \), by the colourings \( C_v \) and \( C'_v \).

---

\(^4\) \( M \) is the positive module-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 \( q = \sum_{i=0}^{k} a_i^7 \), the functions \( f \) and \( g \) are well-defined.

\(^6\) Note that \( m = \frac{3m}{2} \), so \( 7m < 14n \) in the word \( w \).
Lemma 4. There is a minimal grammar $G = (N, \Sigma, R, S)$ for $w_G$ such that, for every $A \in N$, $|\mathcal{D}(A)| \geq 1$ implies that $\mathcal{D}(A)$ is a factor of some $\langle 7i + C_v(i) \rangle_v \#$, $1 \leq i \leq n$, or a factor of some $\langle j \rangle_o$, $1 \leq j \leq 14n$.

Lemma 5. There is a minimal grammar $G$ for $w_G$ such that, for every $i$, $1 \leq i \leq 14n$, there is a nonterminal with derivative $\langle i \rangle_o$ 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 $|u|_* = 196n$, Lemma 4 implies that $|\beta| \geq 196n$, where $\beta$ is the prefix of the compressed string producing $u$. Also by Lemma 4, every $\langle i \rangle_o$ 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 $\beta$ 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 $x \in \{\langle i \rangle_o, \langle i \rangle_v\}$, we can add a new rule $A_x \rightarrow \alpha_x$ with $|\alpha_x| = 3$ and $\mathcal{D}(A_x) = x$ (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 $\beta$ by $7\lceil \frac{k}{2} \rceil \geq 3k$.

We now add rules $\tilde{V}_i \rightarrow \#V_{7i+C_v(i)}\#$, $\tilde{V}_i \rightarrow V_{7i+C_v(i)}\#$, where $\mathcal{D}(V_{7i+C_v(i)}) = \langle 7i + C_v(i) \rangle_v$, and use them, in addition to the rules for the codewords $\langle i \rangle_o$, to obtain a new compressed string from $w_G$ (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 $\tilde{V}_i \rightarrow \tilde{V}_i\#$). Then, we erase all old rules with derivatives $\#(7i + C_v(i)) \# r_i, r_i \# (7i + C_v(i)) \#$. 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 $w_G$ with the rules $\{r_{0,i}, r_{k,i}: 1 \leq i \leq 14n\}$, where $r_{0,i} = D_1 \rightarrow d_1 * d_1$ and $r_{0,1} = V_i \rightarrow x_i * x_i$, $1 \leq i \leq 7$, and $r_{i,j} = D_1 \rightarrow g(i)[1]D_{h(i)}g(i)[1]$ and $r_{i,1} = V_i \rightarrow f(i)[1]V_{h(i)}f(i)[1]$, $8 \leq i \leq 14n$ with $h(i) = \frac{1}{2}(i - M(i), 7)$, $\{\tilde{V}_i \rightarrow \#V_{7i+C_v(i)}\#: 1 \leq i \leq n\}, \{\tilde{V}_i \rightarrow \#\tilde{V}_i: i \in J\}$, for an $J \subseteq \{1, \ldots, n\}$, and with the compressed string $\prod_{i=0}^{6}(\prod_{j=1}^{14n}(D_i V_{M(i+j,14n)}) \prod_{i=1}^{n}(\tilde{V}_i e_i D_{7i-1}) \prod_{i=1}^{n}(\tilde{V}_i D_{7i-2} e_i))$

where for every $i$, $1 \leq i \leq n$, $y_i = \tilde{V}_i$, if $i \in J$ and $y_i = \tilde{V}_i \#$, otherwise, and for every $k$, $1 \leq k \leq m$, $z_k \in \{\tilde{V}_{7j_1-1}, \tilde{V}_{7j_2-1}, \tilde{V}_{7j_3-1} \}$ if $\{j_1, j_2, j_3\} \cap J \neq \emptyset$, $z_k = \tilde{V}_{7j_2-1, j_3} \#$, otherwise.

Proof Sketch. Lemma 5 ensures nonterminals $V_i \rightarrow A$ with $\mathcal{D}(A) = \langle i \rangle_v$. We now replace it by a rule $V_i \rightarrow x_i * x_{i}$ or $V_i \rightarrow f(i)[1]V_{h(i)}f(i)[1]$, as described in the statement of the lemma. If $|\alpha_x| \geq 3$, this is fine, but if $|\alpha_x| = 2$, we have to argue more carefully: we remove $V_i \rightarrow AB$ 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 $\langle i \rangle_o$. Now it only remains to prove that the compressed string can be assumed to have the desired form. If we replace all $\langle i \rangle_v, \langle i \rangle_o, \#(i \rangle_v$ and $\langle i \rangle_v \#$ in $w_G$ 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 $\#(i \rangle_v$ that have been compressed by a single nonterminal and are now compressed by $V_i \#$. This can be repaired by simply adding a rule $\tilde{V}_i \rightarrow \#V_i$, resulting in the set $J$ mentioned in the statement of the lemma.
Lemma 6 allows us to argue similarly as for the reduction from [4]: $\Gamma = \{v_i : i \in \mathcal{I}\}$ is a vertex cover (if $\{v_i, v_j\} \in E$ is not covered, then adding $\overline{V}' \rightarrow \#V'_i$ does not increase the size of the grammar) and $|\Gamma| = f(m, n) + |\Gamma|$, where $f$ is a polynomial. Furthermore, a vertex cover $\Gamma$ translates into a grammar of size $f(m, n) + |\Gamma|$, by setting $\mathcal{I} = \{i : v_i \in \Gamma\}$. Thus, $G$ has a vertex cover $\Gamma$ if and only if there is a grammar $G$ for $w_G$ with $|G| \leq f(m, n) + |\Gamma|$.

\textbf{Theorem 7.} SGP is \textit{NP}-complete, even for alphabets of size 24.

\section{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 $\Phi_{in}(w)$ 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 $\Phi_{in}(w)$ 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

$V_1 = \{(i, j) : 1 \leq i \leq j \leq |w|\}$, $E_1 = \{\{(i_1, j_1), (i_2, j_2)\} : i_1 < i_2 \leq j_1\}$, $V_2 = F_{\geq 2}(w)$, $E_2 = \{\{w[i..j], (i, j)\} : 1 \leq i < j \leq |w|\}$, $V_3 = \{(u, i) : u \in V_2, 0 \leq i \leq |u|\}$, $E_3 = \{(u, (u, i)) : u \in V_2, 0 \leq i \leq |u|\}$.

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 $(j, j + (i - 1)) \in V_1$, $1 \leq j \leq |w| - (i - 1)$, the vertices $\{u \in V_2 : |u| = i\}$ and the vertices $\{(u, j) \in V_3 : |u| = i, 0 \leq j \leq |u|\}$ (see Figure 1 for an illustration).

\textbf{Lemma 8.} Let $w \in \Sigma^*$, $k \geq 1$. 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 $|G| = k - |F_{\geq 2}(w)|$.

\textbf{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 $V_2$-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.

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

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 $\Phi_m(w) = (V, E)$ be defined as follows. Let $V = V_1 \cup V_2 \cup V_3$, where $V_1$ and $V_2$ are defined as for $\Phi_1(w)$, $V_3 = \{(u, 0) : u \in V_2\}$ and $V_4 = \bigcup_{u \in V_2} V_{4,u}$ with $V_{4,u} = \{(u, i,j) : 1 \leq i \leq j \leq |u|, u[i...j] \neq u\}$ 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 $\Phi_1(w)$, $E_3 = \{(u, (u, 0)) : u \in V_2\} \cup \{(u, (u, i,j)) : u \in V_2, (u, i,j) \in V_4,u\}$, $E_4 = \bigcup_{u \in V_2} E_{4,u}$, and, for every $u \in V_2$, $E_{4,u} = \{(u, i,j), (u, i', j')\} \subseteq V_{4,u} : i \leq i' \leq j'$ and $E_5 = \{(u, (v, i,j)) : u, v \in V_2, v[i...j] = u, u \neq v\}$.

Intuitively speaking, $\Phi_m(w)$ 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 $u \in V_2$ 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 $\Phi_m(w)$.

![Figure 2](image-url) Second and third layer of $\Phi_m(abaabaa)$ (vertices $V_1$ and edges $E_1 \cup E_2$ omitted).

**Lemma 9.** Let $w \in \Sigma^*$, $k \geq 1$. There is an independent dominating set $D$ of cardinality $k$ for $\Phi_m(w)$ if and only if there is a grammar $G$ for $w$ with $|G| = k - |F_{\geq 2}(w)|$.

**Proof Sketch.** The correspondence of independent dominating sets $D$ for $\Phi_m(w)$ 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 $|v| \geq 2$ 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 $\Phi(w) \in \{\Phi_1(w), \Phi_m(w)\}$ translates into a grammar for $w$, which, in the following, we will denote by $G(D)$. 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 $G = (N, \Sigma, R, s)$ with $\{S\}(A): A \in N = F$, for some set $F \subseteq F_{\geq 2}(w)$, then we need an independent dominating set $D$ with $(F_{\geq 2}(w) \setminus F) \subseteq D$ and $F \cap D = \emptyset$. Obviously, $D$ is the disjoint union of $(F_{\geq 2}(w) \setminus F)$ and an independent dominating set $D'$ for the graph $H = \Phi(w) \setminus (N[F_{\geq 2}(w) \setminus F] \cup F)$, 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 $F \subseteq F_{\geq 2}(w)$, we define $\text{MinIDS}(w, F) = D_H \cup (F_{\geq 2}(w) \setminus F)$, where $D_H$ is a smallest independent dominating set for $H = \Phi_m(w) \setminus (N[F_{\geq 2}(w) \setminus F] \cup F)$, and $1L\text{-MinIDS}(w, F)$ is defined analogously by using $\Phi_1(w)$ instead of $\Phi_m(w)$. In this way, for any set $F \subseteq F_{\geq 2}$, 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 $F \subseteq F_{\geq 2}(w)$. $\text{MinIDS}(w, F)$ and $1L\text{-MinIDS}(w, F)$ can be computed in time $O(|w|^9)$ and $O(|w|^4)$, respectively. Furthermore, $G(\text{MinIDS}(w, F))$ is a minimal grammar for $w$ and $G(1L\text{-MinIDS}(w, F))$ 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 $F \subseteq F_{\geq 2}(w)$ with $|F| \leq k$ and computing $G(\text{MinIDS}(w, F))$. 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 $O(|w|^{2k+6})$ ($O(|w|^{2k+4})$, resp.).

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

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

This can be proven by reducing from independent set, i.e., a graph $G = ([v_1, v_2, \ldots, v_n], E)$ and $k \in \mathbb{N}$ is transformed into the word $w = \prod_{(v_i, v_j) \in E} (\#v_i \#v_j \#\circ) \prod_{i=1}^n (\#v_i \#\circ)^{n - |N(v_i)|}$, where $N(v_i)$ is the neighbourhood of $v_i$ and every occurrence of $\circ$ stands for a distinct symbol. To see 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 $A_i \rightarrow \#v_i$.

## 5 Exact Exponential Time Algorithms

Computing $G(\text{MinIDS}(w, F))$, for all $F \subseteq F_{\geq 2}(w)$, yields a simple brute-force algorithm with a running time in $O(2^{|w|^2})$. 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 $O(8^{|w|})$. In the following, we shall give more sophisticated exact exponential-time algorithms with running times in $O^*(1.8392^{|w|})$, for the 1-level case, and $O^*(3^{|w|})$, for the multi-level case.

---

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].

9. There are $C_{\lfloor w \rfloor - 1} \leq |w|^{-1}$ ordered binary trees with $|w|$ leaves, where $C_{\lfloor w \rfloor - 1}$ is the $\lfloor (|w| - 1)! \rfloor$th Catalan number, and every ordered tree can be obtained from a binary one by contracting some of its edges.
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 $D(\alpha)$ induced by $\alpha$ is the tuple $(D_G(A_1), \ldots, D_G(A_k))$. 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 $\Gamma_q = \{u_i : 1 \leq i \leq k, |u_i| \geq 2\}$ and let the 1-level grammar $G_q = (N_q, \Sigma, R_q, cs_q)$ be defined by $R_q = \{(A, u) : u \in \Gamma_q\}$, $N_q = \{A_u : u \in \Gamma_q\}$ 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 $\{G_q : q$ is a factorisation of $w\}$ yields an $O^* (2^n)$ 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 $O^*(1.8392^{|w|})$.

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 $cs, D(cs), D(D(cs)), \ldots, w$ (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 $N_i = \{A \in N : (D^*_G(A) \in \Sigma^+ \land (D^*_G)^{-1}(A) \notin \Sigma^+)\}$ and let $L_i : (N \cup \Sigma)^* \rightarrow (N \cup \Sigma)^*$, $1 \leq i \leq d$, be component-wise defined by $L_i(x) = D(x)$, if $x \in N_i$ and $L_i(x) = x$, otherwise.

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

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 $A \rightarrow \alpha$ with $A \in N_d$ can be defined like in the 1-level case, i.e., $p(A) = |cs|_A(|\alpha| - 1) - |\alpha|$, considering that removing this rule and replacing each occurrence of $A$ in $cs$ by $\alpha$ 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 $A \rightarrow \alpha$ with $A \in N_i$ by $p(A) := |(L_{i+1} \circ L_{i+2} \circ \cdots \circ L_d)(cs)|_A(|\alpha| - 1) - |\alpha|$. This allows us to compute the size of a $G$ by $|w| - \sum_{A \in N} p(A)$. The dynamic programming algorithm runs through steps $i = 1, 2, \ldots, d$ and in step $i$, it considers all possibilities for two factorisations $q_{i-1}$ and $q_i$ of $w$ induced by $(L_i \circ L_{i+1} \circ \cdots \circ L_d)(cs)$ and $(L_{i+1} \circ \cdots \circ L_d)(cs)$, respectively (note that this implies $q_{i-1} \subseteq q_i$). 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 $q_{i-1} = (u_1, u_2, \ldots, u_k)$, i.e., for some $j_i$, $0 \leq i \leq k$, with

\[ (f \circ g)(a) = f(g(a)). \]
We conclude this work by deriving some parameterised complexity results. For unexplained concepts of parameterised complexity, we refer to Downey and Fellows [5].

Acknowledgments

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.

References


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