SIIM-TR-A-04-14

SIIM Technical Report

On the Complexity of Optimal Grammar-Based Compression

by

JAN ARPE AND RÜDIGER REISCHUK

Schriftenreihe der Institute für

Informatik/Mathematik

Serie A

December 15, 2004

Universität zu Lübeck

Technisch-Naturwissenschaftliche Fakultät

Email: arpe@tcs.uni-luebeck.de

Phone: +49-451-500-5312

Fax: +49-451-500-5301

On the Complexity of Optimal Grammar-Based Compression

Jan Arpe* Rüdiger Reischuk

Institut für Theoretische Informatik
Universität zu Lübeck, Haus 64
Ratzeburger Allee 160, 23538 Lübeck, Germany

Email: {arpe,reischuk}@tcs.uni-luebeck.de

December 13, 2004

Abstract

The task of grammar-based compression is to find a small context-free grammar generating exactly one given string. We investigate the relationship between grammar-based compression of strings over unbounded and bounded alphabets. Specifically, we show how to transform a grammar for a string over an unbounded alphabet into a grammar for a block coding of that string over a fixed bounded alphabet and vice versa. From these constructions, we obtain asymptotically tight relationships between the minimum grammar sizes for strings and their block codings. Finally, we exploit an improved bound of our construction for *overlap-free* block codings to show that a polynomial time algorithm for approximating the minimum grammar for *binary* strings within a factor of c yields a polynomial time algorithm for approximating the minimum grammar for strings over *arbitrary* alphabets within a factor of $24c + \varepsilon$ (for arbitrary $\varepsilon > 0$). Since the latter problem is known to be NP-hard to approximate within a factor of 8569/8568, we provide a first step towards solving the long standing open question whether minimum grammar-based compression of *binary* strings is NP-complete.

*Supported by DFG research grant Re 672/4.

1 Introduction

Grammar-based compression is a variant of compression by textual substitution, a field pioneered by Lempel and Ziv [16, 17] and further generalized by Storer and Szymanski [14, 15]. Given a string x, the goal of grammar-based compression is to construct a small context-free grammar G_x that generates only the string x. The process of grammar-based compression can in general be split into two parts: first constructing the grammar (the mapping $x \mapsto G_x$ is typically called a *grammar transform*) and then coding the grammar in some compact way. While some investigations concern the latter problem (see for example Kieffer and Yang [3] and Neville-Manning and Witten [9]), most research concentrates on designing efficient grammar transforms, see [1, 3, 5, 8, 9, 17]. Most grammar transforms appearing in the literature have been investigated by Lehman and Shelat [12, 6, 7] in terms of their ability to approximate the smallest possible grammar. A natural measure for the size of such grammars—considered throughout the literature on grammar-based compression—is the sum of the lengths of the right parts of their production rules.

While the primary motivations for data compression are to reduce the size of stored data and to speed up transmissions on networks, the hierarchical structure of grammar-based compression admits some additional interesting applications such as DNA sequence entropy estimators [4], organization of digital libraries [10], and inference of hierarchical structure in natural languages, music, and L-systems [8, 9].

Although grammar-based compression has been successfully applied in several areas, there are many theoretical aspects that deserve to be investigated more deeply. In particular, this concerns the efficient approximability of the *Minimum Grammar Compression (MGC)* problem. The problem is to find, given a string x, a grammar generating exactly x and being as small as possible.

1.1 Known NP-hardness and Inapproximability Results

MGC is NP-complete [13], even when restricted to strings over alphabets of size three. However, it is still unsolved whether it is also NP-hard (or at least not in P unless $\mathsf{P} = \mathsf{NP}$) to construct grammars of minimum size for *binary* strings. We abbreviate the corresponding problem by 2MGC.

Because of the NP-hardness in general, one is interested in at least approximating a minimal grammar. So far, the best known polynomial time approximation algorithms, which have been discovered independently by Charikar et al. [2] and Rytter [11], construct grammars of size at most $O(\log n)$ times the smallest possible size, where n is the length of the string to be compressed.

The largest known lower bound on the approximation ratio that can be achieved by a polynomial time algorithm is 8569/8568 (unless $\mathsf{P} = \mathsf{NP}$).

On the other hand, it is not clear whether the upper bound of $O(\log n)$ is optimal. In particular, for a greedy algorithm belonging to the class of so-called *global* *algorithms* which perform a certain type of grammar transform, the best known lower bound on its approximation ratio is $(5\log 3)/(3\log 5) \approx 1.138$ (this ratio being achieved for a unary string, see [6]). The lowest proven upper bound on the approximation ratio for this algorithm is $O((n/\log n)^{2/3})$ (cf. [6]). An improvement of the upper bound on the approximation ratio to the order of $o(\log n/\log\log n)$ would yield significant progress on the approximation of shortest addition chains, which represents a difficult algebraic challenge. See Sections 3.2 and 3.3 in Lehman [6] for a short survey on addition chains and their connection to grammar-based compression of unary strings.

1.2 New Results

We have investigated how grammar-based compression over arbitrary alphabets and finite alphabets are related. A common way of coding an arbitrary string x in some finite alphabet Σ is to use a block code φ assigning to each symbol appearing in x a distinct code word in Σ^ℓ (for some appropriate block length ℓ). We will inductively construct, given a grammar-compression for x, a grammar-compression for its coding $\varphi(x)$ and vice versa. Both constructions will relate the minimum grammar sizes for a string and its block coding to each other. The bounds we show are asymptotically tight, and in addition to proving this, we also show that tightness holds even for highly compact block codes. These tightness results are obtained by considering code words consisting of shifted blocks such that their preimage is hard to compress.

From the viewpoint of combinatorial optimization, the main research goal in grammar-based compression is to tighten (in)approximability results, especially for finite (and in particular for binary) alphabets. This paper provides a step towards achieving this goal by showing that a superconstant lower bound on the approximation ratio of the general grammar-based compression problem (over arbitrary alphabets) induces a superconstant lower bound on the approximation ratio of the problem restricted to strings over finite alphabets (even of size two). In particular, such a result would imply that MGC when restricted to binary alphabets is not in P unless $\mathsf{P} = \mathsf{NP}$.

There are many reasons for our particular interest in *binary* alphabets: First, all data to be processed digitally usually *is* binary data. Second, binary alphabets are the smallest "usual" alphabets of interest for compression, making their investigation interesting from a structural point of view. Third, all known hardness results yet require larger alphabets, showing the need for a deeper investigation of the binary case.

1.3 Organization of this Paper

In Section 2, we introduce definitions and notations concerning strings, block codes, and grammars and include three basic lemmas. Section 3 is concerned with the relation of grammar sizes for strings and their block codings. The main construction of this work is presented in Section 3.1, while tightness results are presented in Section 3.2. The consequences of our constructions for the (in)approximability of MGC are investigated in Section 4. The paper concludes with Section 5.

2 Preliminaries

Let Γ and Σ be alphabets (i.e., nonempty sets). The *length* of a string $x \in \Gamma^*$ is denoted by $|x|$, and ε denotes the empty string of length 0. For $k \in \mathbb{N}$, the set of strings over Γ of length at most k is denoted by $\Gamma^{\leq k}$.

A string $y \in \Sigma^*$ is a *substring* of x if $x = uyv$ for some strings $u, v \in \Gamma^*$. A substring of length ℓ is also called an *ℓ-block*. If $|u| = k$, then we say that *y appears in position k* in x (note that a string starts in position 0 and ends in position $|x|-1$ with this convention). If $u = \varepsilon$, then y is called a *prefix* of x. A position i, $0 \leq i \leq |x|-1$, is in *ℓ-phase* p in x, $0 \leq p \leq \ell-1$, if $i \bmod \ell = p$. In this case, we say that *y appears in ℓ-phase p in x*. Position i is *ℓ-aligned* and y is an *ℓ-aligned substring* of x if i appears in ℓ-phase 0.

Two strings $x, y \in \Sigma^*$ *overlap* if there exist $u, v, w \in \Sigma^*$ such that $|v| \geq 1$, $x = uv$ and $y = vw$. A homomorphism $\varphi : \Gamma^* \to \Sigma^*$ is an *ℓ-block homomorphism* if $|\varphi(\gamma)| = \ell$ for all $\gamma \in \Gamma$. An *ℓ-block code* is an ℓ-block homomorphism that is injective. It is *overlap-free* if for each pair of distinct symbols $\gamma, \gamma' \in \Gamma$, $\varphi(\gamma)$ and $\varphi(\gamma')$ do not overlap.

In the following we fix some notation concerning formal grammars. A *context-free multi-grammar (CFMG)* $\mathcal{G} = (\Sigma, \mathcal{V}, \mathcal{P}, \mathcal{S})$ consists of a finite *terminal alphabet* Σ, a set $\mathcal{V}$ of *nonterminal symbols* (or *nonterminals*) that is disjoint from Σ, a set of *derivations* $\mathcal{P} \subseteq \mathcal{V} \times (\Sigma \cup \mathcal{V})^*$, and a set of *start symbols* $\mathcal{S} \subseteq \mathcal{V}$. A derivation $(A, \alpha) \in \mathcal{P}$ is also denoted by $A \to_{\mathcal{G}} \alpha$ or $A \to \alpha$ if $\mathcal{G}$ is clear from the context. In this case α is called a *derivative* of A. We naturally extend $\mathcal{P}$ to $(\Sigma \cup \mathcal{V})^* \times (\Sigma \cup \mathcal{V})^*$ and denote its transitive closure by $\mathcal{P}^*$. A string $x \in \Sigma^*$ is *generated by* $A \in \mathcal{V}$ if $\mathcal{P}^*(A) = x$. The *language generated by* $\mathcal{G}$ is $\mathcal{L}(\mathcal{G}) = \{x \in \Sigma^* \mid \exists S \in \mathcal{S} : x \text{ is generated by } S\}$. The *size* $|\mathcal{G}|$ of $\mathcal{G}$ is the sum of the lengths of the derivatives in $\mathcal{G}$.

Definition 1 Let Σ be an alphabet and $X \subseteq \Sigma^*$ a finite set of strings over Σ. Then a CFMG $\mathcal{G}$ is a *grammar for X* if

- it is *deterministic*, i.e., for each $A \in \mathcal{V}$, there exists exactly one derivation $A \to_{\mathcal{G}} \alpha$ for some $\alpha \in (\Sigma \cup \mathcal{V})^*$,
- it is *acyclic*, i.e., $\mathcal{V}$ can be ordered such that the derivative of any nonterminal does not contain any preceding nonterminals, and
- $\mathcal{L}(\mathcal{G}) = X$.

In case $X = \{x\}$, we also say that $\mathcal{G}$ is a grammar for x. The size of a smallest grammar for X is denoted by $m^*(X)$ (or $m^*(x)$ if $X = \{x\}$). The *trivial grammar for* X is $(\Sigma, \mathcal{V}, \mathcal{P}, \mathcal{V})$ with $\mathcal{V} = \{A_x \mid x \in X\}$ and $A_x \to x$ for $x \in X$. Thus, the trivial grammar for X has size $\sum_{x \in X} |x|$. In a grammar $\mathcal{G}$ for X, each string $\alpha \in (\Sigma \cup \mathcal{V})^*$ has a unique *expansion*, i.e., a string $\langle \alpha \rangle \in \Sigma^*$ such that $(\alpha, \langle \alpha \rangle) \in \mathcal{P}^*$. Finally denote by $\|\alpha\|$ the length of α's expansion, i.e., let $\|\alpha\| = |\langle \alpha \rangle|$.

From now on, let Γ be a finite or infinite alphabet and Σ a finite alphabet. For $x \in \Gamma^*$, let Γ_x be the set of symbols appearing in x.

Throughout this paper, let Γ be a countable (finite or infinite) alphabet and Σ be a finite alphabet. If $x \in \Gamma^*$ is a string, let Γ_x be the set of symbols appearing in x.

Lemma 1 *Let $\ell \in \mathbb{N}$ and $X = \{0,1\}^{\leq \ell} \setminus \{\varepsilon\}$. Then there is a grammar of size at most $2^{\ell+2}$ for X.*

Proof. We construct the grammar by induction on ℓ. For $\ell = 1$, there obviously is a grammar of size 2 for X.

For $\ell \geq 2$, we start with the grammar for all strings of length up to $2^{\ell-1}$, which by induction has size at most $2^{\ell+1}$. In that grammar, there is for each $x \in \{0,1\}^{\leq \ell-1}$ a start symbol S_x whose expansion is x. For each $x \in \{0,1\}^{\ell}$, we add a new nonterminal A_x with derivation $A_x \to A_{x[1..k-1]} x[k]$. In total, the size of the constructed grammar is at most $2^{\ell+1} + 2 \cdot 2^{\ell} = 2^{\ell+2}$. □

Lehman [6] provides a lower bound for the grammar size:

Lemma 2 *Let $x \in \Gamma^n$. Then $m^*(x) \in \Omega(\log n)$.*

To bound the size in terms of the number of different substrings of a certain length, we need another lemma from Lehman [6].

Lemma 3 *Let $\mathcal{G}$ be a grammar of size m for $x \in \Gamma^*$. Then for each $k \in \{1, \ldots, |x|\}$, the number of distinct substrings of length k is at most mk.*

3 Block Codes and Grammar Size

3.1 Grammar Sizes for Strings and their Codings

In this section, we are concerned with the following problem. Given a string $x \in \Gamma^*$ and a coding $\varphi(x)$ of x in some fixed alphabet Σ, how are the minimum grammar sizes of x and $\varphi(x)$ related? We investigate this question for ℓ-block codes mapping Γ_x to Σ^*. The final goal is to reduce the case of unbounded alphabets to bounded alphabets by such block codes. In Section 4, we will show that the construction provided in this section leads to some consequences about the relation between the polynomial time approximability of the Minimum Grammar Compression problem for unbounded and for bounded alphabets.

Theorem 1 *Let $x \in \Gamma^*$ and $\varphi : \Gamma_x^* \to \Sigma^*$ be an ℓ-block code. Then the following statements are true:*

(a) From a grammar for x of size $m(x)$ and a grammar for $\varphi(\Gamma_x)$ of size $m(\varphi)$, we can construct a grammar for $\varphi(x)$ of size

$$m(\varphi(x)) \leq m(x) + m(\varphi) .$$

(b) From a grammar for $\varphi(x)$ of size $m(\varphi(x))$, we can construct a grammar for x of size

$$m(x) \leq 2 \cdot \ell \cdot m(\varphi(x)) .$$

(c) If φ is overlap-free, then we can improve the last bound to

$$m(x) \leq 2 \cdot m(\varphi(x)) .$$

All constructions can be done in polynomial time.

Proof. (a) Let $\mathcal{G}_x = (\mathcal{V}_x, \Gamma_x, \mathcal{P}_x, \{S\})$ be a grammar for x and

$$\mathcal{G}_\varphi = (\mathcal{V}_\varphi, \Sigma, \mathcal{P}_\varphi, \{S_\gamma \mid \gamma \in \Gamma_x\})$$

be a grammar for $\varphi(\Gamma_x)$. The idea is to merge the grammars $\mathcal{G}_x$ and $\mathcal{G}_\varphi$ and substitute all terminals γ in $\mathcal{G}_x$ by the corresponding start symbols S_γ from $\mathcal{G}_\varphi$. Formally, we proceed as follows. Without loss of generality we may assume that $\mathcal{V}_x$, $\mathcal{V}_\varphi$, Γ_x, and Σ are pairwise disjoint. Then a grammar for $\varphi(x)$ is given by $\mathcal{G}_{\varphi(x)} = (\mathcal{V}_x \cup \mathcal{V}_\varphi, \Sigma, \mathcal{P}, \{S\})$ with

$$\mathcal{P}(A) = \begin{cases} \mathcal{P}_x(A)|_{\gamma \leftarrow S_\gamma} & \text{if } A \in \mathcal{V}_x, \\ \mathcal{P}_\varphi(A) & \text{if } A \in \mathcal{V}_\varphi, \end{cases}$$

where $\mathcal{P}_x(A)|_{\gamma \leftarrow S_\gamma}$ denotes the string $\mathcal{P}_x(A)$ with all terminal symbols $\gamma \in \Gamma_x$ substituted by nonterminals S_γ. The size of the constructed grammar is $|\mathcal{G}_{\varphi(x)}| = |\mathcal{G}_x| + |\mathcal{G}_\varphi|$.

(b) Let $\mathcal{G}_{\varphi(x)} = (\mathcal{V}_{\varphi(x)}, \Sigma, \mathcal{P}, \{S\})$ be a grammar for $\varphi(x)$. For each $A \in \mathcal{V}_{\varphi(x)}$, $\langle A \rangle$ may appear in ℓ different ℓ-phases $0, \ldots, \ell - 1$ in $\varphi(x)$.

For each $A \in \mathcal{V}_{\varphi(x)}$ and each i with $0 \leq i \leq \min\{\ell - 1, \|A\|\}$, we (uniquely) factorize $\langle A \rangle = u_{A,i} v_{A,i} w_{A,i}$ such that $|u_{A,i}| = i$, $|v_{A,i}| = c \cdot \ell$ for some $c \geq 0$, and $|w_{A,i}| < \ell$. Thus, $v_{A,i}$ is the longest substring of $\langle A \rangle$ that starts in position i and whose length is a multiple of ℓ. In case $\|A\| < i \leq \ell - 1$, we set $u_{A,i} = \langle A \rangle$ and $v_{A,i} = w_{A,i} = \varepsilon$. For each i such that there exists a substring x' of x of length at least two with $v_{A,i} = \varphi(x')$, we introduce a new nonterminal $A^{(i)}$ with $\langle A^{(i)} \rangle = x'$. In this case, we call the pair (A, i) *useful*.

Order the nonterminals of $\mathcal{G}_{\varphi(x)}$ such that no nonterminal appears in a derivative of a preceding nonterminal. In particular, start with a nonterminal whose derivative contains only terminals. We construct the derivations for the nonterminals $A^{(i)}$ inductively according to the just defined order. The following is a description of both

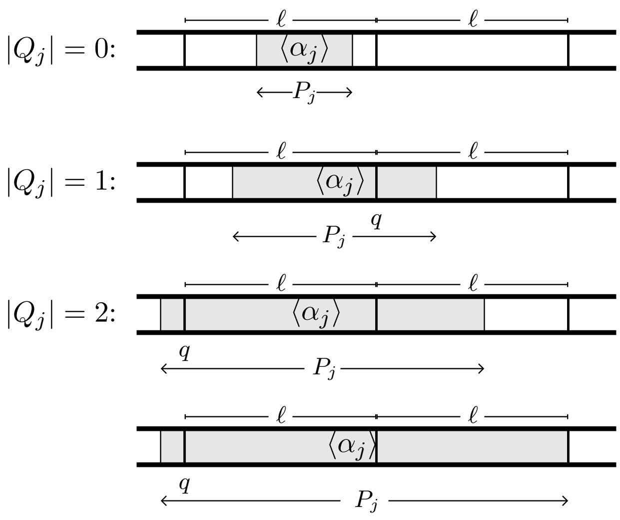

Figure 1: Illustration of cases with $|Q_j| = 0, 1, 2$

the base case and the induction step since the induction hypothesis is only used in Cases 3b, 4a, and 4b below which cannot occur in the base case.

Let (A, i) be a useful pair and let $\alpha = \mathcal{P}(A)$ denote the A's derivative. For each position $j \in \{0, \ldots, |\alpha| - 1\}$, let

$$P_j = \{\|\alpha[0..j-1]\|, \ldots, \|\alpha[0..j]\| - 1\}$$

be the set of positions in $\langle A \rangle$ generated by the j-th symbol in α. Since we are particularly interested in starting positions of ℓ-blocks with offset i generated by α_j, we define

$$Q_j = P_j \cap \{p \mid 0 \leq p \leq \|A\| - \ell \wedge p \equiv i \bmod \ell\} .$$

The derivative of $A^{(i)}$ will be obtained by substituting each symbol $\alpha[j]$ in α by a string β_j of length 0, 1, or 2, depending on the number of ℓ-blocks with offset i in $\langle A \rangle$ generated by $\alpha[j]$. We consider four cases, three of which are illustrated in Fig. 1.

Case 1: $|Q_j| = 0$: In this case, let $\beta_j = \varepsilon$.
Case 2: $|Q_j| = 1$: Let $q \in Q_j$. We define β_j to be the preimage of the ℓ-block starting in position q in $\langle A \rangle$ under φ, i.e., $\beta_j = \varphi^{-1}(\langle A \rangle[q \mathrel{..} q + \ell - 1])$.
Case 3: $|Q_j| = 2$: Let $q = \min Q_j$. We consider two subcases:
Case 3a: $q + 2\ell - 1 \notin P_j$: This means that $\alpha[j]$ generates only one ℓ-block with offset i in $\langle A \rangle$, but also a nonempty prefix of a consecutive ℓ-block. In this case, we proceed similarly to Case 2 and let $\beta_j = \varphi^{-1}(\langle A \rangle[q \mathrel{..} q + 2\ell - 1])$.
Case 3b: $q + 2\ell - 1 \in P_j$: Then α_j generates two ℓ-blocks with offset i in $\langle A \rangle$, but no proper prefix of a consecutive block. We let $\beta_j = \alpha[j]^{(\text{offset}(j))}$, where $\text{offset}(j) = q - \min P_j$. By induction, the new nonterminal $\alpha[j]^{(\text{offset}(j))}$ has already been defined and $\langle \alpha[j]^{(\text{offset}(j))} \rangle = \varphi^{-1}(\langle A \rangle[q \mathrel{..} q + 2\ell - 1])$.
Case 4: $|Q_j| \geq 3$: Let $q_{\min} = \min Q_j$ and $q_{\max} = \max Q_j$. As in Case 3, we consider two subcases:

Case 4a: $q_{\max} + \ell - 1 \notin P_j$: This means that $\alpha[j]$ generates $|Q_j| - 1$ ℓ-blocks with offset i in $\langle A\rangle$ and in addition a nonempty prefix of a consecutive ℓ-block. In this case, we let $\beta_j = \alpha[j]^{(\mathrm{offset}(j))}\varphi^{-1}(\langle A\rangle[q_{\max} \mathinner{..} q_{\max} + \ell - 1])$ with $\mathrm{offset}(j) = q_{\min} - \min P_j$ as in Case 3b.
Case 4b: $q_{\max} + \ell - 1 \in P_j$: Then α_j generates at least two ℓ-blocks with offset i in $\langle A\rangle$, and we let $\beta_j = \alpha[j]^{(\mathrm{offset}(j))}$, where $\mathrm{offset}(j) = q - \min P_j$. By induction, the new nonterminal $\alpha[j]^{(\mathrm{offset}(j))}$ has already been defined and $\langle\alpha[j]^{(\mathrm{offset}(j))}\rangle = \varphi^{-1}(\langle A\rangle[q_{\min} \mathinner{..} q_{\max} + \ell - 1])$.

Now $\langle\beta_0 \ldots \beta_{|\alpha|-1}\rangle = \varphi^{-1}(v_{A,i})$ since the preimage of each ℓ-block with offset i in $\langle A\rangle$ is generated by exactly one string β_j, and all ℓ-blocks whose preimages are generated by some β_j start in pairwise different positions.

The grammar for x is now $\mathcal{G}_x = (\mathcal{V}, \Gamma, \mathcal{P}, \{S^{(0)}\})$ with

$$\mathcal{V} = \{A^{(i)} \mid A \in \mathcal{V}_{\varphi(x)}, i \in \{0, \ldots, \ell - 1\}, (A, i) \text{ useful}\}$$

and $\mathcal{P}$ as defined above. Since there are at most ℓ useful pairs (A, i) for each $A \in \mathcal{V}_{\varphi(x)}$ and since $|\mathcal{P}(A^{(i)})| \le 2|\mathcal{P}(A)|$, we have $|\mathcal{G}_x| \le 2 \cdot \ell \cdot |\mathcal{G}_{\varphi(x)}|$.

(c) Let φ be overlap-free. We prove by contradiction that for each $A \in \mathcal{V}_{\varphi(x)}$, there is at most one i such that (A, i) is useful. Assume that there are i, i', $0 \le i < i' \le \ell - 1$, such that (A, i) and (A, i') are useful. Then $\langle A\rangle$ contains ℓ-blocks $\varphi(\gamma)$ and $\varphi(\gamma')$ starting in positions i and i', respectively. But $\varphi(\gamma)$ clearly occupies position $\ell - 1$ in $\langle A\rangle$ and so does $\varphi(\gamma')$. Consequently, $\varphi(\gamma)$ and $\varphi(\gamma')$ overlap, which is a contradiction. □

Corollary 1 *Let $x \in \Gamma^*$ and $\varphi : \Gamma_x^* \to \Sigma^*$ be an ℓ-block code. Then*

$$m^*(\varphi(x)) \le m^*(x) + m^*(\varphi(\Gamma_x)) \tag{1}$$
$$m^*(x) \le 2 \cdot \ell \cdot m^*(\varphi(x)) . \tag{2}$$

Unfortunately, if Γ is infinite, then $|\Gamma_x|$ may grow (even linearly) with $|x|$. Consequently, $m^*(\varphi(\Gamma_x))$ may also grow with $|x|$. However, $|\Gamma_x| \le m^*(x)$ is a trivial upper bound, and since the trivial grammar for $\varphi(\Gamma_x)$ has size $\ell \cdot |\Gamma_x|$, $m^*(\varphi(\Gamma_x)) \le \ell \cdot |\Gamma_x| \le \ell \cdot m^*(x)$, hence by (1),

$$m^*(\varphi(x)) \le (\ell + 1) \cdot m^*(x) . \tag{3}$$

In order for an ℓ-block code $\varphi : \Gamma_x^* \to \Sigma^*$ to exist, $\ell \ge \lceil \log_{|\Sigma|} |\Gamma_x| \rceil$ is required, but also sufficient.

Corollary 2 *Let $x \in \Gamma^*$, $\ell = \lceil \log_2 |\Gamma_x| \rceil$ and $\varphi : \Gamma_x^* \to \{0,1\}^*$ be an ℓ-block code. Then*

$$\frac{m^*(x)}{2\lceil \log_2 m^*(x) \rceil} \le m^*(\varphi(x)) \le m^*(x)(\lceil \log_2 m^*(x) \rceil + 1) .$$

Proof. The first inequality follows from (2) and the fact that $m^*(x) \ge |\Gamma_x|$. The second inequality is a direct consequence of the same fact and (3). □

3.2 Tightness of Bounds

Next we show that for arbitrary block codes, (2) is tight up to a constant factor. As we have seen, the factor ℓ stems from the fact that for a nonterminal A, a subword of $\langle A\rangle$ of length ℓ may occur in up to ℓ different "phases". This can for example be forced by using a code of shifted code words as we will now demonstrate.

Theorem 2 *For infinitely many $\ell \in \mathbb{N}$ (more precisely: for each even ℓ), there exist a string $x \in \Gamma^*$ and an ℓ-block monomorphism $\varphi : \Gamma_x^* \to \{0,1\}^*$ such that*

$$m^*(x) \geq \frac{1}{24} \cdot \ell \cdot m^*(\varphi(x)) .$$

Proof. Let ℓ be even and $k = \ell/2$. We work with the alphabet

$$\Gamma = \{b_{ij} \mid 0 \leq i \leq k-1, 0 \leq j \leq k-1\}$$

of size k^2 and define $\varphi : \Gamma \to \{0,1\}^{2k}$ by

$$\varphi(b_{ij}) = 0^i 1 0^j 1 0^{2k-2-i-j} .$$

Then

$$\varphi(b_{i0}b_{i1}\ldots b_{i,k-1})0^i = 0^i\varphi(b_{00}b_{01}\ldots b_{0,k-1}) . \tag{4}$$

Now consider the string

$$x = b_{00}b_{01}\ldots b_{0,k-1}\ b_{10}b_{11}\ldots b_{1,k-1}\ldots\ldots b_{k-1,0}b_{k-1,1}\ldots b_{k-1,k-1}$$

and its image $\varphi(x)$. Since x contains k^2 different symbols, the size of any grammar for x is at least k^2, thus $m^*(x) \geq k^2$. We will show that $\varphi(x)$ can be generated by a grammar of size $12k-10$. First we construct nonterminals A_i generating all strings 0^i, $2 \leq i \leq 2k-2$ by letting $A_2 \to 00$, $A_{i+1} \to A_i 0$. This requires $4k-6$ derivative symbols. Next we introduce a nonterminal generating $\varphi(b_{00}b_{01}\ldots b_{0,k-2})\ 10^{k-1}1$ with a corresponding derivative of length $4k-2$:

$$B \to 11A_{2k-2}\ 101A_{2k-3}\ 1A_2 1A_{2k-4}\ \ldots\ 1A_{k-2}1A_k\ 1A_{k-1}1 .$$

Now we can construct the blocks $\varphi(b_{i0}b_{i1}\ldots b_{i,k-1})$ from the derivation $B_i \to A_i B A_{k-i-1}$ for $0 \leq i \leq k-1$, where $A_0 = \varepsilon$ and $A_1 = 0$, so we get another $3k-2$ grammar symbols on right sides (in case $i \in \{0, k-1\}$, we only have 2 symbols). Finally, we define the start derivation by $S \to B_0 B_1 \ldots B_{k-1}$. In total, the resulting grammar has size $12k-10$.

Altogether, $m^*(x) \leq k^2$ and $\ell \cdot m^*(\varphi(x)) \leq 24k^2$ (since $\ell = 2k$), proving the claim. □

Example 1 Let us illustrate the constructions from the proof by explicitly considering the case $k = 3$. We have

$$\begin{array}{rl}
\varphi(b_{00}b_{01}b_{02}) = & \quad\;\; 1\ 1\ 0\ 0\ 0\ 0 \,|\, 1\ 0\ 1\ 0\ 0\ 0 \,|\, 1\ 0\ 0\ 1\ 0\ 0 \\
\varphi(b_{10}b_{11}b_{12}) = & \quad 0\ 1\ 1\ 0\ 0\ 0 \,|\, 0\ 1\ 0\ 1\ 0\ 0 \,|\, 0\ 1\ 0\ 0\ 1\ 0 \\
\varphi(b_{20}b_{21}b_{22}) = & 0\ 0\ 1\ 1\ 0\ 0 \,|\, 0\ 0\ 1\ 0\ 1\ 0 \,|\, 0\ 0\ 1\ 0\ 0\ 1
\end{array}$$

where spaces and |-symbols have been inserted for better readability. In addition, the alignment of the strings shows the structure of "shifted blocks". We have $x = b_{00}b_{01}b_{02}b_{10}b_{11}b_{12}b_{20}b_{21}b_{22}$, and $\varphi(x)$ is the concatenation of the three rows above. The constructed grammar for $\varphi(x)$ contains the following derivations:

$$\begin{array}{lcllcl}
S & \to & B_0B_1B_2 & \langle S\rangle & = & \varphi(x) \\
B_0 & \to & BA_2 & \langle B_0\rangle & = & \varphi(b_{00}b_{01}b_{02}) \\
B_1 & \to & 0B0 & \langle B_1\rangle & = & \varphi(b_{10}b_{11}b_{12}) \\
B_2 & \to & A_2B & \langle B_2\rangle & = & \varphi(b_{20}b_{21}b_{22}) \\
B & \to & 11A_4101A_31A_21 & \langle B\rangle & = & \varphi(b_{00}b_{01})1001 \\
A_4 & \to & A_30 & \langle A_4\rangle & = & 0000 \\
A_3 & \to & A_20 & \langle A_3\rangle & = & 000 \\
A_2 & \to & 00 & \langle A_2\rangle & = & 00
\end{array}$$

The size of the grammar is $26 = 12 \cdot 3 - 10$.

A drawback of the code construction in the previous theorem is that the symbols of Γ are not encoded very compactly: an alphabet of size k^2 is encoded by binary strings of length $2k$. In the following, we will show how to obtain the same bound as in Theorem 2 for highly compact codes, i.e., for codes φ with block length $\log_2 |\Gamma|$.

Theorem 3 *For infinitely many $\ell \in \mathbb{N}$, there exist an $x \in \Gamma^*$ with $|\Gamma_x| = 2^\ell$ and an ℓ-block code $\varphi : \Gamma_x^* \to \{0,1\}^*$ such that*

(a) $m^*(x) \geq \ell \cdot 2^{\ell-1} - \ell$,

(b) $m^*(\varphi(x)) \leq 2^{\ell+2} + 2\ell - 1$, and thus

(c) $m^*(x) \in \Theta(\ell \cdot m^*(\varphi(x)))$.

Proof. The idea is similar as in the proof of Theorem 2. Let $\Gamma = \{b_0, \ldots, b_{2^\ell-1}\}$, and let $\varphi(b_i)$ be the representation of the number i as a binary string of length ℓ (i.e., $\varphi(b_i) = a_{i,\ell-1} \ldots a_{i,0}$ such that $i = \sum_{j=0}^{\ell-1} a_{i,j} \cdot 2^j$). Instead of constructing x and then considering $\varphi(x)$, we start with a string $y \in \{0,1\}^*$ and then derive x as the preimage of y under φ. Let $s : \{0,1\}^+ \to \{0,1\}^*$ be the *left shift map* that shifts a string one position to the left (and "forgets" the leftmost position), i.e., $s(x_1 \ldots x_n) = x_2 \ldots x_n$. Now let $z = \varphi(b_0) \ldots \varphi(b_{2^\ell-1})$ and define

$$y = z\ s(z)0\ s^2(z)0^2\ \ldots\ s^{\ell-1}(z)0^{\ell-1}\,.$$

Since $|y| = \ell^2 \cdot 2^\ell$ is an integer multiple of ℓ, there exists a (unique) preimage x of y under φ, i.e., $\varphi(x) = y$. Let us investigate $m^*(x)$ first.

Claim 1 *The number of substrings of length 2ℓ in z is $\ell \cdot 2^\ell - 2\ell + 1$.*

Proof of the claim. We show that all $\ell \cdot 2^\ell - 2\ell + 1$ substrings of length 2ℓ of z are pairwise different. Consider two substrings s and t of length 2ℓ appearing in positions i and j in z, respectively, with $i < j$. If $\ell \mid j - i$, then s and t cannot be equal since otherwise they would contain an ℓ-block that appears in two different ℓ-aligned positions contradicting the definition of z. Moreover, if s and t differ in at least one of the first ℓ positions, then they are certainly different. Now assume that i and j are in different ℓ-phases and that the first ℓ symbols of s and t coincide, i.e., $s = rs'$ and $t = rt'$ for some strings $r, s', t' \in \{0,1\}^\ell$. Our goal is to show that $s' \neq t'$. We do this by showing that the ℓ-blocks following the occurrences of r in z are pairwise different. Consider the occurrence of rs' in position i, and denote $\hat\imath = i \bmod \ell$. Split each of r and s' into two parts $r^{(0)} = r_0 \ldots r_{\ell-\hat\imath-1}$, $r^{(1)} = r_{\ell-\hat\imath} \ldots r_{\ell-1}$, $s'^{(0)} = s'_0 \ldots s'_{\ell-\hat\imath-1}$, and $s'^{(1)} = s'_{\ell-\hat\imath} \ldots s'_{\ell-1}$, so that $r^{(1)}$ and $s'^{(1)}$ appear in consecutive ℓ-aligned positions. Let us first consider the case that $r^{(0)} \neq 2^{\ell-\hat\imath} - 2$, where (also in the following) we interpret $r^{(0)}$ as a binary number, i.e., $b_0 \ldots b_{k-1} = \sum_{j=0}^{k-1} b_j 2^{k-1-j}$.

If $r^{(0)} + 2 \neq 2^{\ell-\hat\imath}$, the situation is as follows:

$$\ldots \mid \underbrace{\overbrace{r_0 \ldots r_{\ell-\hat\imath-1}}^{r^{(0)}} \,||\, \overbrace{r_{\ell-\hat\imath} \ldots r_{\ell-1}}^{r^{(1)}}}_{r} \mid \underbrace{\overbrace{s'_0 \ldots s'_{\ell-\hat\imath-1}}^{s'^{(0)}=r^{(0)}+1} \,||\, \overbrace{s'_{\ell-\hat\imath} \ldots s'_{\ell-1}}^{s'^{(1)}=r^{(1)}}}_{s'} \mid \overbrace{\ \ldots\ }^{r^{(0)}+2} \,||\, \ldots$$

We have marked borders between 2ℓ-aligned 2ℓ-blocks by $||$ and borders bounding the ℓ-blocks r and s' by $|$. The numbers $r^{(0)} + 1$ and $r^{(0)} + 2$ are to be understood modulo $2^{\ell-\hat\imath-1}$. Consequently, r and s' coincide in the rightmost $\hat\imath$ positions $\ell - \hat\imath, \ldots, \ell - 1$ and differ in position $\ell - \hat\imath - 1$.

Now consider the case that $r^{(0)} + 2 = 2^{\ell-\hat\imath}$. Then the situation looks like this:

$$\ldots \mid \underbrace{\overbrace{1 \ldots 10}^{r^{(0)}} \,||\, \overbrace{r_{\ell-\hat\imath} \ldots r_{\ell-1}}^{r^{(1)}}}_{r} \mid \underbrace{\overbrace{1 \ldots 1}^{s'^{(0)}=r^{(0)}+1} \,||\, \overbrace{s'_{\ell-\hat\imath} \ldots s'_{\ell-1}}^{s'^{(1)}=r^{(1)}+1}}_{s'} \mid \overbrace{0 \ldots 0}^{r^{(0)}+2} \,||\, \ldots$$

Thus r and s' differ in the rightmost position $\ell - 1$. Consequently, if $s' = t'$, then we must either have

(i) $\hat\jmath = 0$ and $r + 2 \neq 2^\ell$ (where $\hat\jmath = j \bmod \ell$) or

(ii) $\hat\jmath \neq \hat\imath$ with $r_0 \ldots r_{\ell-\hat\jmath-1} + 2 = 2^{\ell-\hat\jmath}$.

In case (i), $s' = r + 1$. Moreover, the incrementation of r by 1 yields a carry bit in position $\ell - \hat\imath - 1$, so $s'^{(1)} = 1 \ldots 1 = 2^{\hat\imath} - 1$ and thus $s' = 2^\ell - 1$, contradicting $r + 2 \neq 2^\ell$.

Case (ii) is also impossible since whenever $r_0 \ldots r_{\ell-\hat\jmath-1} + 2 = 2^{\ell-\hat\jmath}$, then $r = \underbrace{1 \ldots 1}_{\ell-\hat\jmath-1} 0 r'$ for some suitable r', contradicting $\hat\imath \neq \hat\jmath$. $\square$

Now we complete the proof of the theorem. Consider the set of ℓ-aligned 2ℓ-blocks in y: it contains the ℓ-aligned 2ℓ-blocks of z, $s(z)$, …, and $s^{\ell-1}(z)$. But these are exactly *all* 2ℓ-blocks of z, so the number of ℓ-aligned 2ℓ-blocks in y is at least $\ell \cdot 2^\ell - 2\ell + 1$ by Claim 1. Now the preimage of y under φ equals the sequence of the preimages of the ℓ-aligned ℓ-blocks in y, and each such ℓ-block in y corresponds to a single symbol in x. Whenever two such ℓ-blocks are different, then of course their preimages must also be different. Thus the number of 2-blocks in x is as large as the number of ℓ-aligned 2ℓ-blocks in y, i.e., $\ell \cdot 2^\ell - 2\ell + 1$. By Lemma 3, $m^*(x) \geq (\ell \cdot 2^\ell - 2\ell + 1)/2 \geq \ell \cdot 2^{\ell-1} - \ell$.

Finally, we construct a grammar for $\varphi(x)$ of size $2^{\ell+2} + 2\ell - 1$, which will finish the proof. By Lemma 1, there exists a grammar of size $2^{\ell+2}$ for the set of all nonempty strings of lengths at most ℓ over $\{0,1\}$. Let $A(w)$ denote the nonterminal with $\mathcal{P}(A(w)) = w$ from this grammar for $w \in \{0,1\}^*$ with $1 \leq |w| \leq \ell$. Let $k \in \{0, \ldots, \ell-1\}$. Since $s^k(z) = 0^{\ell-k}\varphi(b_1)\ldots\varphi(b_{2^\ell-1})$, $s^k(z)$ is generated by the nonterminal B_k with the derivation $B_k \to A(0^{\ell-k})A(\varphi(b_1))\ldots A(\varphi(b_{2^\ell-1}))$ whose size is 2^ℓ. Now let

$$S \to B_0 B_1 A(0) B_2 A(00) B_3 A(0^3) \ldots B_{\ell-1} A(0^{\ell-1})$$

be the start derivation of the grammar to be constructed. It clearly generates y and has size $2\ell - 1$. The total size of the grammar is $2^{\ell+2} + 2\ell - 1$. □

4 Approximability for Bounded and Unbounded Alphabets

Let us now investigate the consequences of Theorem 1 for the approximability of the Minimum Grammar Compression problem. First we use a special overlap-free code φ and show that the grammar sizes for x and $\varphi(x)$ can only differ by constant factors.

Theorem 4 *Let $x \in \Gamma^*$ and $\ell = 2\lceil \log_2(|\Gamma_x| + 1)\rceil + 2$. Then there exists an ℓ-block code $\varphi : \Gamma_x^* \to \{0,1\}^*$ such that the following statements are true:*

(a) From any grammar for x of size $m(x)$, we can construct a grammar for $\varphi(x)$ of size

$$m(\varphi(x)) \leq (12 + o(1))m(x) .$$

More precisely, for all $\varepsilon > 0$, there exists $n \in \mathbb{N}$ such that if $|x| \geq n$, then from any grammar for x of size $m(x)$, we can construct a grammar for $\varphi(x)$ of size

$$m(\varphi(x)) \leq (12 + \varepsilon)m(x) .$$

(b) From any grammar for $\varphi(x)$ of size $m(\varphi(x))$, we can construct a grammar for x of size

$$m(x) \leq 2m(\varphi(x)) .$$

Proof. Let $\Gamma_x = \{\gamma_1, \ldots, \gamma_k\}$ ($k = |\Gamma_x|$) and $n = \lceil \log_2(k+1) \rceil$. Define

$$\varphi(\gamma_i) = 0^n 1 b(i) 1 ,$$

where $b(i)$ denotes the binary representation of $i \in \{1, \ldots, k\}$ as an n-block. Clearly, the resulting code is an overlap-free ℓ-block code for $\ell = 2n + 2$. Together with Part (c) of Theorem 1, this proves (b), since we have shown how to construct an overlap-free block code. It remains to show (a). This will be done by constructing a grammar for the set of code blocks $\varphi(\Gamma_x)$. As a consequence of Lemma 1, it is possible to generate all n-blocks over $\{0, 1\}$ by a grammar of 2^{n+2}. Let B_i denote the nonterminal with $\langle B_i \rangle = b(i)$ for $1 \le i \le k$. Next we introduce the nonterminal Z with derivation $Z \to 0^n 1$ (this could be done more efficiently, but is enough for our purposes). Finally, for each $i \in \{1, \ldots, k\}$, we introduce a start symbol with derivation $S_i \to Z B_i 1$. In total, the resulting grammar has size

$$\begin{aligned} m(\varphi(\Gamma_x)) &= 2^{n+2} \ + \ n+1 \ + \ 3k \\ &= 2^{\lceil \log_2(k+1) \rceil + 2} + \lceil \log_2(k+1) \rceil + 1 + 3k \\ &\le 2^{(\log_2 k)+3} + 3k + \lceil \log_2(k+1) \rceil + 1 \\ &\le 11k + \lceil \log_2(k+1) \rceil + 1 . \end{aligned}$$

Applying $k = |\Gamma_x| \le m(x)$, we obtain $m(\varphi(\Gamma_x)) \le 11m(x) + o(m(x))$. By Part (a) of Theorem 1, from a grammar of size $m(x)$ for x, we can construct a grammar for $\varphi(x)$ of size

$$\begin{aligned} m(\varphi(x)) &\le m(x) + m(\varphi(\Gamma_x)) \\ &\le 12m(x) + \lceil \log_2(m(x)+1) \rceil + 1 \\ &= (12 + o(1))m(x) . \end{aligned}$$

□

Corollary 3 *A polynomial time factor c-approximation algorithm for* 2MGC *yields a factor* $(24c + \varepsilon)$*-approximation for* MGC *(for arbitrarily small* $\varepsilon > 0$*).*

Proof. Let there be a polynomial time algorithm A that, on input $y \in \{0,1\}^*$, outputs a grammar $\mathcal{G}$ for y of size at most $c \cdot m^*(y)$. Let $x \in \Gamma^*$, and let φ be the block code from Theorem 4. Let $\mathcal{G}_{\varphi(x)}$ be the grammar output by A on input $\varphi(x)$, and let $\mathcal{G}_x$ be a grammar for x of size $|\mathcal{G}_x| \le 2|\mathcal{G}_{\varphi(x)}|$ which exists by Part (b) of Theorem 4. We then have

$$\begin{aligned} |\mathcal{G}_x| &\le 2|\mathcal{G}_{\varphi(x)}| \\ &\le 2 \cdot c \cdot m^*(\varphi(x)) \\ &\le 2 \cdot c \cdot (12m^*(x) + \log_2 m^*(x) + 2) \\ &\le (24c + \frac{2c(\log_2 m^*(x) + 2)}{m^*(x)}) \cdot m^*(x) . \end{aligned}$$

Now let $\varepsilon > 0$. Since $m^*(x) \geq d \log_2 |x|$ for some constant $d > 0$ by Lemma 2, there exists $n \in \mathbb{N}$ such that if $|x| \geq n$,

$$\frac{2c(\log_2 m^*(x) + 2)}{m^*(x)} \leq \frac{2c(\log_2(d \log_2 n) + 2)}{\log_2 n} < \varepsilon$$

and hence $|\mathcal{G}_x| \leq (24c + \varepsilon)m^*(x)$. $\square$

The Corollary implies that if the Minimum Grammar Compression problem cannot be approximated within a constant factor for arbitrary strings, then it cannot be approximated within a constant factor for binary strings either.

Thus, if we were able to show that the general problem does not admit any constant factor approximation in polynomial time, then it would immediately follow that the same is true for the binary case. In particular, such a result would imply that binary grammar-based compression is not in P (unless $\mathsf{P} = \mathsf{NP}$). Moreover, it could yield a major step towards showing NP-hardness of binary grammar-based compression. Note that it would already suffice to show a lower bound of $24 + \varepsilon$ for the approximation ratio of the general problem in order to obtain an inapproximability result for the binary case.

5 Conclusion

We have shown asymptotically tight results on the relationship between the size of grammars for a string x over some possibly infinite alphabet Γ and its block coding $\varphi(x)$ over some finite alphabet Σ. Moreover, by using overlap-free block codings, dependencies between approximability results for grammar-based compression over unbounded and over finite alphabets have been obtained.

One goal for future research is to prove a superconstant lower bound on the (polynomial time computable) approximation ratio for MGC (unless $\mathsf{P} = \mathsf{NP}$) or at least a lower bound of $24 + \varepsilon$ for some $\varepsilon > 0$ in order to show inapproximability of 2MGC.

A differently flavored task of practical impact is the following. Given a finite set $X \subset \Gamma^*$, construct a block code $\varphi : \Gamma_X^* \to \Sigma^*$ (where $\Gamma_X = \cup_{x \in X} \Gamma_x$) for which the optimum grammar-based compression of $\varphi(X)$ can be achieved. To our knowledge, this task has not been investigated in this context yet.

Acknowledgments

The authors thank Bodo Manthey for valid discussions and helpful suggestions.

References

[1] Alberto Apostolico and Stefano Lonardi. Some Theory and Practice of Greedy Off-line Textual Substitution. In *Data Compression Conference, DCC 1998,*

March 30-April 1, 1998, Snowbird, Utah, USA, pages 119–128. IEEE Computer Society, 1998.

[2] Moses Charikar, Eric Lehman, Ding Liu, Rina Panigrahy, Manoj Prabhakaran, April Rasala, Amit Sahai, and Abhi Shelat. Approximating the Smallest Grammar: Kolmogorov Complexity in Natural Models. In *Proceedings of the 34th Annual ACM Symposium on Theory of Computing, May 19-21, 2002, Montréal, Québec, Canada*, pages 792–801. ACM Press, 2002.

[3] John C. Kieffer and En-hui Yang. Grammar Based Codes: A New Class pf Universal Lossless Source Codes. *IEEE Trans. Inform. Theory*, IT-46(3):737–754, May 2000.

[4] J. Kevin Lanctot, Ming Li, and En-hui Yang. Estimating DNA Sequence Entropy. In *Proceedings of the Eleventh Annual ACM-SIAM Symposium on Discrete Algorithms, January 9-11, 2000, San Francisco, CA, USA*, pages 409–418. ACM/SIAM, 2000.

[5] N. Jesper Larsson and Alistair Moffat. Offline Dictionary-Based Compression. *Proc. IEEE*, 88(11):1722–1732, November 2000.

[6] Eric Lehman. *Approximation Algorithms for Grammar-Based Data Compression*. PhD thesis, Dep. of Electrical Engineering and Computer Science, Mass. Inst. of Techn., 2002.

[7] Eric Lehman and Abhi Shelat. Approximation Algorithms for Grammar-Based Compression. In *Proceedings of the Thirteenth Annual ACM-SIAM Symposium on Discrete Algorithms, January 6-8, 2002, San Francisco, CA, USA*, pages 205–212. ACM/SIAM, 2002.

[8] Craig G. Nevill-Manning. *Inferring Sequential Structure*. PhD thesis, Department of Computer Science, University of Waikato, Hamilton, New Zealand, 1996.

[9] Craig G. Nevill-Manning and Ian H. Witten. Compression and explanation using hierarchical grammars. *Computer Journal*, 40(2/3):103–116, 1997.

[10] Craig G. Nevill-Manning, Ian H. Witten, and Gordon W. Paynter. Browsing in digital libraries: A phrase-based approach. In *Proceedings of the 2nd ACM International Conference on Digital Libraries, July 25-28, 1997, Philadelphia, PA, USA*, pages 230–236. ACM, 1997.

[11] Wojciech Rytter. Application of Lempel-Ziv Factorization to the Approximation of Grammar-Based Compression. *Theoret. Comput. Sci.*, 302(1-3):211–222, June 2003. Preliminary version in Alberto Apostolico and Masayuki Takeda, editors, *Combinatorial Pattern Matching, 13th Annual Symposium, CPM 2002, Fukuoka, Japan, July 3-5, 2002, Proceedings*, volume 2372 of *Lecture Notes in Computer Science*, pages 20-31. Springer, 2002.

[12] Abhi Shelat. Evaluating Grammar-Based Compression Algorithms. Master's thesis, Department of Electrical Engineering and Computer Science, Mass. Inst. Tech., August 2001.

[13] James A. Storer. NP-completeness results concerning data compression. Technical Report 234, Dep. of Electrical Engineering and Computer Science, Princeton Univ., Princeton, N.J., 1977.

[14] James A. Storer and Thomas G. Szymanski. The Macro Model for Data Compression (Extended Abstract). In *Proceedings of the Tenth Annual ACM Symposium on Theory of Computing, May 1-3, 1978, San Diego, California, USA*, pages 30–39, 1978.

[15] James A. Storer and Thomas G. Szymanski. Data Compression via Textual Substitution. *JACM*, 29(4):928–951, October 1982.

[16] Jacob Ziv and Abraham Lempel. A Universal Algorithm for Sequential Data Compression. *IEEE Trans. Inform. Theory*, IT-23(3), May 1977.

[17] Jacob Ziv and Abraham Lempel. Compression of Individual Sequences via Variable-Rate Coding. *IEEE Trans. Inform. Theory*, IT-24(5):530–536, September 1978.