Approximation Algorithms for Grammar-Based Compression

Eric Lehman e_lehman@mit.edu
Abhi Shelat abhi@mit.edu
MIT Laboratory for Computer Science
200 Technology Square
Cambridge, MA 02141

Abstract

Several recently-proposed data compression algorithms are based on the idea of representing a string by a context-free grammar. Most of these algorithms are known to be asymptotically optimal with respect to a stationary ergodic source and to achieve a low redundancy rate. However, such results do not reveal how effectively these algorithms exploit the grammar-model itself; that is, are the compressed strings produced as small as possible? We address this issue by analyzing the *approximation ratio* of several algorithms, that is, the maximum ratio between the size of the generated grammar and the smallest possible grammar over all inputs. On the negative side, we show that every polynomial-time grammar-compression algorithm has approximation ratio at least $\frac{8569}{8568}$ unless P = NP. Moreover, achieving an approximation ratio of $o(\log n / \log\log n)$ would require progress on an algebraic problem in a well-studied area. We then upper and lower bound approximation ratios for the following four previously-proposed grammar-based compression algorithms: Sequential, Bisection, Greedy, and LZ78, each of which employs a distinct approach to compression. These results seem to indicate that there is much room to improve grammar-based compression algorithms.

1 Introduction

Grammar-based data compression aims to succinctly represent an input string by a context-free grammar generating only that string. For example, one might represent the string:

abcabccabcabcbcabccabacabbbcabcc
ababcabccabcabcbcabccabacabcaba

by the context-free grammar:

$$\begin{aligned} S &\rightarrow TbUTVa \\ T &\rightarrow aUVcUaV \\ U &\rightarrow bVcV \\ V &\rightarrow cab \end{aligned}$$

This grammar can then be translated into a bit string using a standard technique such as arithmetic encoding. Grammar-based data compression was first proposed explicitly by Kieffer and Yang [6] and Nevill-Manning [9], but is closely related to some earlier "macro-based" schemes proposed by Storer [10].

In addition to achieving competitive compression rates [1, 7, 9], grammar-based algorithms are attractive for several reasons. A string encoded as a grammar remains relatively comprehensible in compressed form. This eases implementation of algorithms such as pattern matching that operate directly on the compressed form for efficiency. Furthermore, this comprehensibility allows one to use grammar-based compression to extract information about patterns in the original string. For example, grammar-based compression has been used to identify patterns in DNA sequences, English text, and musical scores [9].

Several grammar-based compression algorithms have been proposed. Nevill-Manning [9] devised the Sequitur algorithm which incrementally builds a grammar in a single pass through the input string. This procedure was subsequently improved by Kieffer and Yang [6] to what we refer to here as the Sequential algorithm. The same authors employed a completely different approach to generating a compact grammar for a given string in their Bisection algorithm. This procedure partitions the input into halves, then quarters, then eighths, etc. and creates a nonterminal in the grammar for each distinct substring generated in this way. Bisection was subsequently generalized to MPM [7] in order to exploit multi-way and incomplete partitioning. De Marcken [4] presented a complex multi-

pass algorithm that emphasizes avoiding local minima. Apostolico and Lonardi [1] proposed a greedy algorithm (hereafter called GREEDY) in which rules are added in a steepest-descent fashion. Finally, even though it predates the idea of grammar-based compression, the output of the well-known LZ78 algorithm [14] can also be interpreted as a grammar. (In contrast, the output of LZ77 [13] has no natural interpretation as a grammar.)

Traditional analysis of a compression algorithm first concentrates on showing that the algorithm is universal with respect to stationary ergodic sources; that is, it asymptotically approaches the optimal compression rate. Then, as a refinement, one might bound the redundancy, which measures how quickly an algorithm approaches that optimum. All of the above algorithms except for SEQUITUR and de Marcken's are known to be universal.

We measure grammar-based compression algorithms by a simpler standard: how large is the output grammar relative to the smallest grammar? More precisely, a grammar-based compression algorithm has *approximation ratio* $\rho(n)$ if for every input string of length n, the algorithm outputs a grammar at most $\rho(n)$ times larger than the smallest grammar for that string. Here the size of a grammar is defined to be the total number of symbols on the right sides of all rules. One might prefer to define the size of a grammar to be the length of its encoding in bits, since this is the ultimate measure of compression. However, our coarser definition has the merit of simplicity, and its imprecision is dwarfed by the approximation ratios of the algorithms we analyze.

At a high level, studying approximation ratios gives insight into how fully one can exploit the grammar model of compression. Fully exploiting a simple compression model, such as run-length encoding, is easy. On the other hand, it is impossible to make full use of a powerful model in which a string is represented by an encoding of a Turing machine that prints it. At a practical level, approximation ratio analysis allows one to differentiate between grammar-based compressors with no assumptions about the source of input. (After all, many files compressed in practice are not generated by stationary ergodic sources.) From a theoretical perspective, grammar-based compression is an elegant combinatorial optimization problem. Context-free grammars are fundamental to computer science, but have rarely been studied from an optimization perspective. Furthermore, this extends the study of approximation algorithms to hierarchical objects (grammars) as opposed to "flat" objects (graphs, CNF-formulas, etc.) This is a significant shift since many real-world problems have a hierarchical nature, but standard approximation techniques such as linear and semidefinite programming are not easily applied to this new domain.

We begin by showing that the grammar model of compression can not be exploited to the absolute maximum: every polynomial-time algorithm has approximation ratio at least $\frac{8569}{8568}$ unless P = NP. We also show that achieving an approximation ratio of $o(\log n/\log\log n)$ will require progress on an algebraic problem in a well-studied area. We then switch to analyzing upper and lower bounds on the approximation ratios for a variety of previously-proposed compressors. Our results are summarized in the table below. Here n is the length of the input string.

	Approximation Ratio	
Algorithm	Upper Bound	Lower Bound
LZW	$O((n/\log n)^{2/3})$	$\Omega(n^{2/3}/\log n)$
BISECTION	$O((n/\log n)^{1/2})$	$\Omega(n^{1/2}/\log n)$
SEQUENTIAL	$O((n/\log n)^{3/4})$	$\Omega(n^{1/3})$
GREEDY	$O((n/\log n)^{2/3})$	$> 1.37\ldots$

The bounds for LZ78 hold for some variants, including LZW [11]. Results for MPM mirror those for BISECTION. The lower bound for SEQUENTIAL also holds for SEQUITUR. We were unable to analyze de Marcken's algorithm.

While significant uncertainties remain, the startling result is that *not one of the most-studied grammar-based compressors has a good approximation ratio.* The best proved approximation ratio is $O(\sqrt{n/\log n})$ for BISECTION. Only the GREEDY algorithm holds promise of being much better. On the other hand, the difficulties that trip up the other algorithms do not look at all fundamental. There seems to be much potential for progress in this area.

2 Preliminaries

A *grammar* G is a 4-tuple $(\Sigma, \Gamma, S, \Delta)$. Here Σ is a finite alphabet whose elements are called *terminals*, Γ is a disjoint set whose elements are called *nonterminals*, and $S \in \Gamma$ is a special nonterminal called the *start symbol*. All other nonterminals are called *secondary*. In general, the word *symbol* refers to any terminal or nonterminal. In this paper, terminals are lowercase, nonterminals are uppercase, and strings of symbols are lowercase Greek. We always use σ to denote the input string, $n = |\sigma|$ for its length, m for the size of a particular grammar for σ, and m^* for the size of the smallest grammar.

The last component of a grammar is a set of rules Δ of the form $T \to \alpha$, where $T \in \Gamma$ is a nonterminal and $\alpha \in (\Sigma \cup \Gamma)^*$ is a string of symbols referred to as the *definition* of T. In the grammars we consider, each nonterminal T has exactly one rule $T \to \alpha$ in Δ. Furthermore, all grammars are acyclic; that is, there

exists an ordering of the nonterminals Γ such that each nonterminal precedes all nonterminals in its definition. These properties guarantee that a grammar accepts exactly one finite-length string.

The *expansion* of a string of symbols α is obtained by iteratively replacing each nonterminal in α by its definition until only terminals remain. In particular, the string represented by a grammar is the expansion of its start symbol. The size of a grammar G is the total number of symbols in all definitions:

$$\sum_{T \to \alpha \in \Delta} |\alpha|$$

Note that a grammar of size m can be encoded using $O(m \log m)$ bits in a straightforward way. This observation bounds the imprecision introduced by our use of grammar size as a measure of compression performance.

Finally, we use several notational conventions to compactly express strings. The symbol | represents a terminal that appears only once in a string. When | is used several times in the same string, each appearance represents a different symbol. For example, $a \mid bb \mid cc$ contains five distinct symbols. Product notation is used to indicate repetition, and parentheses are used for grouping. For example $(ab)^5 = ababababab$ and $\prod_{i=1}^{3} ab^i \mid = ab \mid abb \mid abbb \mid$.

3 Hardness

We establish the hardness of the grammar compression problem in two ways. First, we show that approximating the size of the smallest grammar to within a small constant factor is NP-hard.

THEOREM 3.1. *There is no polynomial-time grammar-based compressor with approximation ratio less than* $\frac{8569}{8568}$ *unless* $P = NP$.

Proof. We use a reduction from a restricted form of vertex cover based closely on an argument by Storer [10]. Let $G = (V, E)$ be a graph with maximum degree three. Map this graph to the following string over an alphabet that includes a symbol corresponding to each vertex $v_i \in V$:

$$\prod_{v_i \in V} (\#v_i \mid v_i\# \mid)^2 \prod_{v_i \in V} \#v_i\# \mid \prod_{(v_i, v_j) \in E} \#v_i\#v_j\# \mid$$

This string has length $16|V|+6|E|$, and the smallest grammar for it has size $15|V| + 3|E| + k$, where k is the size of the minimum vertex cover of G. (A minimum-size grammar may as well contain rules for all strings of the form $\#v_i$ and $v_i\#$. The set of all strings $\#v_i\#$ for which there are rules defines a vertex cover of G.) However, the minimum vertex cover for this restricted family of graphs is known to be hard to approximate below a ratio of 145/144 [2], and so it is hard to approximate the smallest grammar for this string below the claimed threshold. □

Now we demonstrate the hardness of grammar-based data compression in an alternative sense: a compressor with a low approximation ratio would imply progress on an apparently difficult algebraic problem in a well-studied area.

Let x be a real number, and let $k_1, k_2, \ldots, k_p$ be positive integers. How many multiplications are required to compute $x^{k_1}, x^{k_2}, \ldots, x^{k_p}$? (Algorithms that use other operations are ruled out. Thus, more precisely, what is the shortest addition chain containing all of $k_1, k_2, \ldots, k_p$? The theory of addition chains is extensive [8].) This problem is known to be NP-hard if the integers k_i are given in binary [5]. However, even if they are written in unary, apparently no polynomial-time algorithm with approximation ratio $o(\log n / \log \log n)$ is known, where $n = \sum k_i$. The following theorem states that improving the approximation ratio for grammar-based compression beyond this threshold is at least as difficult.

THEOREM 3.2. *Let* $T = \{k_1, \ldots k_p\}$ *be a set of distinct positive integers, and define*

$$\sigma = x^{k_1} \mid x^{k_2} \mid \ldots \mid x^{k_p}.$$

Then the following relationship holds, where l^* *is the minimum number of multiplications required to compute all of* $x^{k_1}, x^{k_2}, \ldots, x^{k_p}$, *and* m^* *is the size of the smallest grammar for string* σ*:*

$$l^* \leq m^* \leq 4l^*$$

The idea of the proof is that a grammar for σ can be read as an algorithm for computing $x^{k_1}, x^{k_2}, \ldots, x^{k_p}$ and vice-versa.

4 Approximation Algorithms

In this section, we establish upper and lower bounds on the approximation ratios of four grammar-based data compression algorithms: LZ78, BISECTION, SEQUENTIAL, and GREEDY.

4.1 LZ78

The well-known LZ78 [14] algorithm can be regarded as a grammar-based compressor. The procedure

works as follows. Begin with an empty grammar. Make a single left-to-right pass through the input string. At each step, find the shortest prefix of the unprocessed portion that is not the expansion of a secondary nonterminal. This prefix is either a single terminal a or else expressible as Xa where X is an existing nonterminal and a is a terminal. Define a new nonterminal, either $Y \to a$ or $Y \to Xa$, and append this new nonterminal to the end of the start rule.

For example, on input 001010110101011011111, LZ78 defines secondary rules as follows:

$$\begin{array}{lcllcllcl} X_1 & \to & 0 & X_4 & \to & 1 & X_7 & \to & X_21 \\ X_2 & \to & X_11 & X_5 & \to & X_40 & X_8 & \to & X_71 \\ X_3 & \to & X_20 & X_6 & \to & X_51 & X_9 & \to & X_41 \end{array}$$

The start rule is $S \to X_1X_2X_3X_4X_5X_6X_7X_8X_9$.

The next two theorems provide closely-matching upper and lower bounds on the approximation ratio of LZ78.

THEOREM 4.1. *The approximation ratio of* LZ78 *is* $\Omega(n^{2/3}/\log(n))$.

Proof. Consider the input string: $0^{k(k+1)/2}1(0^k1)^{(k+1)^2}$. LZ78 first generates nonterminals with expansions $0, 00, \ldots, 0^k$, and then nonterminals with expansions of the form 0^i10^j for all $0 \le i, j \le k$. Therefore, the size of the grammar produced by LZ78 is $\Omega(k^2)$, while there exists a grammar of size $O(\log k)$. The theorem follows since $k = \Theta(n^{1/3})$. □

The upper bound on the approximation ratio for LZ78 relies on the following fundamental lemma that relates the complexity of a string to the size of its grammar.

LEMMA 4.1. *If a string σ has a grammar of size m, then σ contains at most ml distinct substrings of length l.*

Proof. Let G be a grammar for σ of size m. We will first construct a set Ω consisting at most ml string of length l. Then we prove that every length-l substring of σ is a member of this set. For each rule $A \to \alpha$ in G, add the following strings to Ω:

1. For each terminal in α, add the length l string in the expansion of α which begins at this terminal.

2. For each nonterminal in α, add the $l-1$ strings of length l in the expansion of α that begin with between 1 and $l-1$ characters in the expansion of the nonterminal.

In order to bound the size of Ω, note that we add at most $l-1$ strings to Ω for each character in α. There are at most m characters in all of the rules in G. Therefore, summing over all of the rules in G implies that $|\Omega| \le m(l-1) < ml$.

Now let s be an arbitrary length-l substring of σ. In order to prove that $s \in \Omega$, order the rules of G by increasing expansion length and find the first rule whose expansion entirely contains s. Within this rule, s either begins at a terminal or inside the expansion of a nonterminal. In the former case, Ω will contain s. In the latter case, since s was too big to fit entirely within this smaller nonterminal, it must be the case that s has only between 1 and $l-1$ symbols within the nonterminal's expansion. Therefore, s is again in Ω.□

THEOREM 4.2. *The approximation ratio of* LZ78 *is* $O\left((n/\log n)^{2/3}\right)$.

Proof. Let m^* be the size of the smallest grammar for input string σ of length n, and let $S \to X_1X_2 \ldots X_m$ be the start rule generated by LZ78. Note that the size of the LZ78 grammar is at most $3m$ since every other nonterminal is defined by a rule of the form $X_i \to X_j\alpha$ or simply $X_i \to \alpha$. Hence, it suffices to upper bound m. By the definition of LZ78, each nonterminal X_i expands to a distinct string and is used exactly once in the start rule. List these nonterminals X_i by increasing expansion length. Lemma 4.1 states that a string representable by a small grammar contains few distinct, short substrings. In particular, the lemma implies that each nonterminal among the first group of m^* in this list has an expansion with length at least 1, each nonterminal in the next group of $2m^*$ has an expansion with length at least 2, and so forth. Suppose that after repeating this argument k times, not enough nonterminals remain in the list to form another complete group; that is:

$$m \quad < \quad m^* + 2m^* + 3m^* + \cdots + km^* + (k+1)m^*$$

This implies that $m = O(k^2m^*)$. On the other hand, the sum of the expansion lengths of all the grouped nonterminals is at most n:

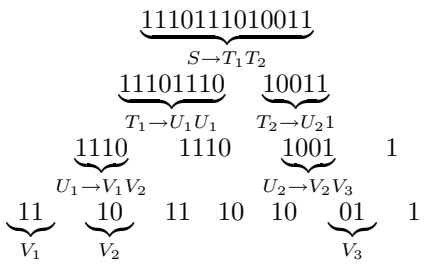
$$m^* + 2 \cdot 2m^* + 3 \cdot 3m^* + \cdots + k \cdot km^* \quad \le \quad n$$

This implies that $k = O((n/m^*)^{1/3})$. Substituting gives $m = O((n/m^*)^{2/3}m^*)$. Finally, noting that m^* is $\Omega(\log n)$ for every input string gives $m = O(((n/\log n)^{2/3}m^*)$ from which the theorem follows. □

4.2 The Bisection Algorithm

Kieffer and Yang introduced the BISECTION algorithm in [7]. This procedure works on an input string σ as follows. Select the largest integer k such that $2^k < |\sigma|$. Partition σ into two substrings with lengths 2^k and $|\sigma| - 2^k$. Repeat this partitioning process recursively on each substring of length greater than one. Create a nonterminal for every distinct string of length greater than one generated during this process. Each such nonterminal can then be defined by a rule with exactly two symbols on the right.

As an example, consider the string $\sigma = 1110111010011$. The recursive partitioning and association of a nonterminal with each distinct substring generated is shown below:



THEOREM 4.3. *The approximation ratio of* BISECTION *is* $O(\sqrt{n/\log n})$ *and* $\Omega(\sqrt{n}/\log n)$.

Proof. For the lower bound, let σ be the string $1(0^{2^k}1)^{2^k-1}$, which has total length $n = 2^{2k}$. For example, if $k = 2$, then σ is 1000 0100 0010 0001. Bisecting σ k times gives 2^k distinct substrings of length 2^k, thereby generating a grammar of size $\Omega(2^k)$. Since there is a grammar of size $O(k)$ for σ, BISECTION has an approximation ratio of $\Omega\left(\frac{2^k}{k}\right) = \Omega\left(\frac{\sqrt{n}}{\log n}\right)$.

For the upper bound, let σ be an arbitrary string, let G be BISECTION's grammar for σ, and let m^* be the size of the smallest grammar for σ. The size of G is at most twice the number of distinct substrings generated during the recursive partitioning process, so it suffices to upper bound the latter quantity. Let k be the largest integer such that $2^k < |\sigma|$. Then this process generates $\lfloor n/2^i \rfloor$ strings of length 2^i for $1 \le i \le k$ together with up to k additional strings of various lengths. Using Lemma 4.1 for a better bound on the number of distinct, short strings, we obtain the following upper bound on the size of the BISECTION grammar:

$$\begin{aligned} |G| &= O\left(k + \sum_{i=1}^{\frac{1}{2}(k-\log k)} m^* 2^i + \sum_{i=\frac{1}{2}(k-\log k)}^{k} \left\lfloor \frac{n}{2^i} \right\rfloor\right) \\ &= O(\log n) + O\left(m^* \sqrt{\frac{n}{\log n}}\right) + O\left(\sqrt{n \log n}\right) \\ &= O\left(m^* \sqrt{\frac{n}{\log n}}\right) \end{aligned}$$

□

BISECTION was generalized to an algorithm called MPM [7], which permits the recursive partitioning to split more than two ways and to terminate early. For reasonable parameters, performance bounds are the same as for BISECTION.

4.3 Sequential Algorithm

Nevill-Manning and Witten introduced the SEQUITUR grammar compression algorithm in [9]. Kieffer and Yang [6] subsequently offered an improved algorithm, which we refer to here as SEQUENTIAL. SEQUENTIAL works as follows. Begin with an empty grammar, and make a single left-to-right pass through the input string. At each step, find the longest prefix of the unprocessed portion of the input that matches the expansion of a secondary nonterminal, and append that nonterminal to the start rule. Otherwise, if no prefix matches the expansion of a secondary nonterminal, append the first terminal in the unprocessed portion to the start rule. In either case, if the last pair of symbols in the start rule already occurs at some non-overlapping position in the grammar, then replace both occurrences by a new nonterminal whose definition is that pair. Finally, if some nonterminal is used only once after this substitution, then replace it by its definition, and delete the corresponding rule.

As an example, consider the input string $\sigma = x \mid xx \mid xxxx \mid xxxxxxxx$. In each of the first six steps, a single terminal is appended to the start rule, and the grammar becomes $S \to x \mid xx \mid x$. No secondary rules have been created, because every non-overlapping pair of symbols occurs only once in this prefix. However, when the next x is appended to the start rule, there are two copies of the substring xx. Therefore the rule $R_1 \to xx$ is added to the grammar, and both occurrences of xx are replaced by R_1. The grammar is now:

$$\begin{aligned} S &\to x \mid R_1 \mid R_1 \\ R_1 &\to xx \end{aligned}$$

Because the expansion of R_1 is now a prefix of the unprocessed part of σ, the next step consumes xx and

appends R_1 to S. During the next few steps, the start rule expands to $S \to x \mid R_1 \mid R_1R_1 \mid R_1R_1$. At this point, the pair R_1R_1 appears twice, and so a new rule is $R_2 \to R_1R_1$ is added and applied. SEQUENTIAL eventually produces the following grammar for σ:

$$\begin{array}{lcl} S & \to & x \mid R_1 \mid R_2 \mid R_2R_2 \\ R_1 & \to & xx \\ R_2 & \to & R_1R_1 \end{array}$$

THEOREM 4.4. *The approximation ratio of* SEQUENTIAL *is* $\Omega(n^{1/3})$.

Proof. The idea is to exploit SEQUENTIAL's tendency to match the longest rule in order to persuade it to represent the same string in many different ways. Define δ_i to be the string 0^i10^{k-i}. Consider the input string $\alpha \mid \beta^{k/2}$ where

$$\begin{array}{lcl} \alpha & = & 0^k \mid 0^k \mid \delta_0 \mid \delta_0 \mid \delta_1 \mid \delta_1 \mid \ldots \mid \delta_k \mid \delta_k \\ \beta & = & \delta_k\delta_k\ \delta_k\delta_{k-1}\ \delta_k\delta_{k-2}\ \delta_k\delta_{k-3}\ \ldots\ \delta_k\delta_{k/2}\ 0^{k-1} \end{array}$$

After α is processed, the grammar contains a nonterminal for 0^k, a nonterminal for each string δ_i, and other nonterminals for shorter strings that will never be used again. The first copy of β is parsed as the string is written above. However, the final 0^{k-1} is combined with the leading zero in the second copy of β and read as a single nonterminal. With this leading zero already processed, SEQUENTIAL parses the second copy of β completely differently:

$$\delta_{k-1}\delta_{k-1}\ \delta_{k-1}\delta_{k-2}\ \delta_{k-1}\delta_{k-3}\ \ldots\ \delta_{k-1}\delta_{k/2-1}\ 0^{k-2}$$

Now the final 0^{k-2} is combined with the two leading zeroes in the third copy of β and read as a single nonterminal. Consequently, SEQUENTIAL parses the third copy of β in yet another way. In general, each copy of β adds k symbols to the start rule. No pair of nonterminals appears twice, and so no new rules are created. Since there are $k/2$ copies of β, the final grammar has size $\Omega(k^2)$. However, there is a succinct grammar for this string of size $\Theta(k)$. The theorem follows since $k = \Theta(n^{1/3})$. □

To upper bound SEQUENTIAL's performance, we rely on a property of SEQUENTIAL's output. Kieffer and Yang [7] show that SEQUENTIAL produces an *irreducible* grammar; that is, one in which (1) all non-overlapping pairs of symbols are distinct, (2) every nonterminal appears at least twice, and (3) no two distinct symbols have the same expansion. The upper bound on the approximation ratio of SEQUENTIAL is a corollary of the following theorem.

THEOREM 4.5. *Every irreducible grammar for a string is* $O((n/\log n)^{3/4})$ *times larger than the smallest grammar for that string.*

Proof. Let σ be a string of length n, and let G be an irreducible grammar of size m for σ. Each non-overlapping pair of symbols in G represents some substring of σ. Since G is irreducible, two pairs of symbols in G can represent the same substring only if they partition that substring in different ways. A string of length l can be partitioned in at most $l-1$ ways. Using Lemma 4.1, we conclude that at most $(l-1)lm^*$ pairs of symbols in G represent substrings of length l in σ. Moreover, the total length of all expansions of these pairs is at least $(l-1)l^2m^*$.

One can see that there exist at least $m/3$ non-overlapping pairs of adjacent symbols in G. As in the proof of Theorem 4.2, we list these pairs by increasing expansion length. From the argument above, each pair among the first group of $2(2-1)m^*$ in the list has expansion length of at least 2. Similarly, each pair among the next $2 \cdot 3m^*$ has an expansion of length at least 3. Continue to group these pairs into sets of size $3 \cdot 4m^*, \ldots, (k-1)km^*$ until there are not enough pairs to form the next group. At this point, we know that

$$2m^* + 2 \cdot 3m^* + \cdots + (k-1) \cdot km^* \le m/3$$

Therefore, $m = O(k^3m^*)$.

By Lemma 9.33 in [6], the expansions of all rules in the irreducible grammar G have a combined length at most $2n$. Therefore

$$2 \cdot 2(1)m^* + 3 \cdot 3(2)m^* + \cdots + k \cdot k(k-1)m^* \quad \le \quad 2n$$

This inequality implies that $k = O\left((\frac{n}{m^*})^{1/4}\right)$. Substituting into the first bound gives $m = O((n/m^*)^{3/4}m^*)$. The theorem follows since $m^* = \Omega(\log n)$. □

Unfortunately, analyzing the properties of irreducible grammars is not sufficient to substantially improve this upper bound, because there exists an irreducible grammar of size $\Omega(n^{2/3})$ even for the string a^n. Thus, substantially improving this upper bound will require detailed analysis of the inner workings of SEQUENTIAL. However, one need only consider binary strings; a string over a large alphabet can always be mapped to a string over the binary alphabet such that the approximation ratio changes by only about a $\log n$ factor.

4.4 Greedy

Apostolico and Lonardi [1] considered greedy algorithms for grammar-based data compression. Their idea

is to begin with a grammar where the definition of the start symbol is the entire input string. Then one repeatedly adds the rule that decreases the size of the grammar as much as possible. Each rule is added by making a pass through the string from left to right and replacing each occurrence of the definition of the rule by its nonterminal. GREEDY terminates when no rule can be added without enlarging the grammar. For example, one might begin with the grammar:

$$S \rightarrow 111111100000011101111$$

GREEDY first adds $T \rightarrow 111$, since this rule decreases the size of the grammar as much as possible:

$$\begin{aligned} S &\rightarrow TT1000000T0T1 \\ T &\rightarrow 111 \end{aligned}$$

The rules $U \rightarrow 000$ and $V \rightarrow T1$ are added in turn, and the final grammar is:

$$\begin{aligned} S &\rightarrow TVUUT0V & U &\rightarrow 000 \\ T &\rightarrow 111 & V &\rightarrow T1 \end{aligned}$$

THEOREM 4.6. *The approximation ratio for* GREEDY *is* $O((n/\log n)^{2/3})$ *and not less than* $\frac{5 \log 3}{3 \log 5} = 1.137\ldots$.

The upper bound argument is similar to the one for SEQUENTIAL. A detailed proof that GREEDY produces an irreducible grammar is presented in the full version of this paper. However, we can also show that grammars produced by GREEDY are such that no non-overlapping pairs of adjacent symbols can expand to the same string. As a consequence, we can extract at least $m/3$ non-overlapping pairs of adjacent symbols, *all with distinct expansions*. In the analysis of SEQUENTIAL we could only assume that for a pair with an expansion of length k, there were at most $k-1$ other pairs with the same expansion. This extra fact allows us to tighten the argument.

For the lower bound, we analyze how GREEDY handles an input string σ that consists of 5^{2^k} copies of the same symbol. GREEDY creates a grammar of size $5 \cdot 2^k$. Roughly, GREEDY creates nonterminals for x^{5^i} for all $1 \leq i \leq 2^k$. However, asymptotically it is more efficient to create nonterminals for x^{3^i} for all $1 \leq i \leq 2^k \log_3 5$ and then to exploit these to form σ using an insignificant number of additional symbols. More precisely, one can show that there exists a grammar of size $(3 \log_3 5 + o(1)) \cdot 2^k$. The $1.137\ldots$ lower bound follows.

5 Conclusion

The present work only scratches the surface of this area. The problem of designing a time and space efficient grammar-based compression algorithm with provably good approximation ratio is both theoretically fascinating and practically motivated.

Beyond this, there are many natural hierarchical optimization problems in the same vein. For example, one can imagine a grammar-like scheme for compressing images in which nonterminals represent rectangular subimages. There is also a circuit design problem in which a set of input signals together with a collection of subsets of the signals is specified. The problem is to determine the smallest circuit consisting entirely of AND gates that computes the AND of each subset of input signals in the collection. As far as we know, both of these problems are open.

The authors would like to thank Madhu Sudan and Mohammad Mahdian for helpful discussions and Yevgeniy Dodis and Amit Sahai for suggesting the approximation perspective on grammar-based compression.

References

[1] A. Apostolico and S. Lonardi. Some Theory and Practice of Greedy Off-Line Textual Substitution. DCC 1998, pp 119-128.

[2] P. Berman and M. Karpinski. On Some Tighter Inapproximability Results, Further Improvements. ECCC 1998.

[3] D. Bleichenbacher. Efficiency and Security of Cryptosystems Based on Number Theory. PhD thesis, Swiss Federal Institute of Technology, 1996.

[4] C. de Marcken. The Unsupervised Acquisition of a Lexicon from Continuous Speech. MIT AI Memo 1558. November 1995.

[5] P. Downey, B. Leong, and R. Sethi. Computing Sequences with Addition Chains. SIAM Journal on Computing, 10(3):638–646, August 1981.

[6] J. C. Kieffer and E. Yang. Grammar-Based Codes: a New Class of Universal Lossless Source Codes. IEEE Transactions on Information Theory, vol. 46 (2000), pp. 737–754.

[7] J. C. Kieffer, E. Yang, G. J. Nelson, P. Cosman. Universal Lossless Compression via Multilevel Pattern Matching. IEEE Transactions on Information Theory, vol. 46 (2000), pp. 1227–1245.

[8] D. Knuth. Seminumerical Algorithms. Addison-Wesley, 1981, pp. 441–462.

[9] C. Nevill-Manning. Inferring Sequential Structure. PhD thesis, University of Waikato, 1996.

[10] J. Storer. Data Compression: Methods and Complexity Issues. PhD Thesis, Princeton University, 1979.

[11] T. A. Welch. A Technique for High Performance Data Compression. IEEE Computer, vol. 17 (June 1984), pp. 8–19.

[12] E. Yang and J. C. Kieffer. Efficient Universal Lossless Data Compression Algorithms Based on a Greedy Sequential Grammar Transform. IEEE Transactions on Information Theory, vol. 46 (2000), pp. 755–777.

[13] J. Ziv and A. Lempel. A Universal Algorithm for Sequential Data Compression. IEEE Transactions on Information Theory, vol. 23 (1977), pp. 337–343.

[14] J. Ziv and A. Lempel. Compression of Individual Sequences via Variable-Rate Coding. IEEE Transactions on Information Theory, vol. 24 (1978), pp. 530–536.