The Smallest Grammar Problem

Moses Charikar, Eric Lehman, Ding Liu, Rina Panigrahy, Manoj Prabhakaran, Amit Sahai, and abhi shelat

***Abstract*—This paper addresses the *smallest grammar problem*: What is the *smallest* context-free grammar that generates exactly one given string σ?**

This is a natural question about a fundamental object connected to many fields such as data compression, Kolmogorov complexity, pattern identification, and addition chains.

Due to the problem's inherent complexity, our objective is to find an approximation algorithm which finds a small grammar for the input string. We focus attention on the *approximation ratio* of the algorithm (and implicitly, the worst case behavior) to establish provable performance guarantees and to address shortcomings in the classical measure of *redundancy* in the literature.

Our first results are concern the hardness of approximating the smallest grammar problem. Most notably, we show that every efficient algorithm for the smallest grammar problem has approximation ratio at least $8569/8568$ unless P $=$ NP. We then bound approximation ratios for several of the best known grammar-based compression algorithms, including LZ78, BISECTION, SEQUENTIAL, LONGEST MATCH, GREEDY, and RE-PAIR. Among these, the best upper bound we show is $O(n^{1/2})$. We finish by presenting two novel algorithms with exponentially better ratios of $O(\log^3 n)$ and $O(\log(n/m^*))$, where m^* is the size of the smallest grammar for that input. The latter algorithm highlights a connection between grammar-based compression and LZ77.

***Index Terms*—Approximation algorithm, data compression, hardness of approximation, LONGEST MATCH, LZ77, LZ78, multilevel pattern matching (MPM), RE-PAIR, SEQUITUR, smallest grammar problem.**

I. INTRODUCTION

THIS paper addresses the *smallest grammar problem*; namely, what is the smallest context-free grammar that generates exactly one given string? For example, the smallest context-free grammar generating the string a rose is a rose is a rose is as follows:

$$S \to BBA$$
$$A \to \text{a rose}$$
$$B \to A \text{ is}.$$

The size of a grammar is defined to be the total number of symbols on the right sides of all rules. In the example above, the grammar has size 14. Because the decision version of this problem is NP-complete, our objective is to find an approximation algorithm which finds a grammar that generates the given string and is not much larger than the smallest such grammar.

This elegant problem has considerable interest in its own right: it is a simple question about a basic mathematical object, context-free grammars. By virtue of this simplicity and the wide applicability of context-free grammars, the smallest grammar problem has interesting connections to many fields of study, including data compression, Kolmogorov complexity, pattern identification, and approximation algorithms.

A. Data Compression

Instead of storing a long string, one can store a small grammar that generates the string. The original string can be easily reconstructed from the grammar when needed. Many data compression procedures use this idea, and therefore amount to approximation algorithms for the smallest grammar problem [1]–[9]. Most of these procedures are analyzed in detail in Section VI.

Empirical results indicate that the grammar-based approach to compression is competitive with other techniques in practice [4], [9], [6], [7], [10], [11], and some grammar-based compressors are known to be asymptotically optimal on input strings generated by finite-state sources. But in Section VI we show that, surprisingly, many of the best known compressors of this type can fail dramatically; that is, there exist input strings generated by small grammars for which these compressors produce large grammars. Consequently, they turn out not to be very effective approximation algorithms for the smallest grammar problem.

B. Complexity

The size of the smallest context-free grammar generating a given string is also a natural, but more tractable variant of Kolmogorov complexity [12]. The Kolmogorov complexity of a string σ is the length of the shortest pair (M, x) where M is a Turing machine description, x is a string, and M outputs σ on input x. This Turing machine model for representing strings is too powerful to be exploited effectively; in general, the Kolmogorov complexity of a string is incomputable. However, weakening the string representation model from Turing machines to context-free grammars reduces the complexity of the problem from the realm of undecidability to mere intractability. Moreover, we show that one can efficiently approximate the "grammar complexity" of a string.

Manuscript received May 21, 2002; revised February 2, 2005. The material in this paper was presented in part at the Symposium on Discrete Algorithms (SODA '02), San Francisco, CA, January 2002 and the Symposium on the Theory of Computing (STOC '02), Las Vegas, NV, January 2002.

M. Charikar, D. Liu, and M. Prabhakaran are with the Department of Computer Science, Princeton University, Princeton, NJ 08540 USA (e-mail: moses@cs.princeton.edu; dingliu@cs.princeton.edu; mp@cs.princeton.edu).

E. Lehman and a. shelat are with CSAIL, Massachusetts Institute of Technology, Cambridge, MA 02139 USA (e-mail: e_lehman@csail.mit.edu; abhi@csail.mit.edu).

R. Panigrahy is with the Department of Computer Science, Stanford University, Stanford, CA 94305 USA (e-mail: rinap@cs.stanford.edu).

A. Sahai is with the Department of Computer Science, University of California, Los Angeles, Los Angeles, CA 90095 USA (e-mail: sahai@cs.ucla.edu).

Communicated by M. J. Weinberger, Associate Editor for Source Coding.

Digital Object Identifier 10.1109/TIT.2005.850116

0018-9448/$20.00 © 2005 IEEE

Authorized licensed use limited to: Trial User - Warsaw University (Uniwersytet Warszawski). Downloaded on August 09,2023 at 11:58:59 UTC from IEEE Xplore. Restrictions apply.

C. Pattern Recognition

The smallest grammar problem is also relevant to identifying important patterns in a string, since such patterns naturally correspond to nonterminals in a compact grammar. In fact, an original and continuing motivation for work on the problem was to identify regularities in DNA sequences [6], [8]. (Interestingly, [8] espouses the goal of determining the entropy of DNA. This amounts to upper-bounding the Kolmogorov complexity of a human being.) In addition, smallest grammar algorithms have been used to highlight patterns in musical scores [13] and uncover properties of language from example texts [9]. All this is possible because a string represented by a context-free grammar remains relatively comprehensible. This comprehensibility is an important attraction of grammar-based compression relative to otherwise competitive compression schemes. For example, the best pattern matching algorithm that operates on a string compressed as a grammar is asymptotically faster than the equivalent for the well-known LZ77 compression format [14].

D. Hierarchical Approximation

Finally, work on the smallest grammar problem qualitatively extends the study of approximation algorithms. Prior work on approximation algorithms has focused on "flat" objects such as graphs, Boolean formulas in conjunctive normal form, bins, weights, etc. In contrast, context-free grammars as well as many real-world problems such as circuit design and image compression have a hierarchical nature. Moreover, standard approximation techniques such as linear and semidefinite programming are not easily transferred to this new domain.

II. Previous Work

The smallest grammar problem was articulated explicitly by two groups of authors at about the same time. Nevill-Manning and Witten stated the problem and proposed the SEQUITUR algorithm as a solution [6], [13]. Their main focus was on extracting patterns from DNA sequences, musical scores, and even the Church of Latter-Day Saints genealogical database, although they evaluated SEQUITUR as a compression algorithm as well.

The other group, consisting of Kieffer, Yang, Nelson, and Cosman, approached the smallest grammar problem from a traditional data compression perspective [5], [4], [3]. First, they presented some deep theoretical results on the impossibility of having a "best" compressor under a certain type of grammar compression model for infinite length strings [15]. Then, they presented a host of practical algorithms including BISECTION, multilevel pattern matching (MPM), and LONGEST MATCH. Furthermore, they gave an algorithm, which we refer to as SEQUENTIAL, in the same spirit as SEQUITUR, but with significant defects removed. All of these algorithms are described and analyzed in Section VI. Interestingly, on inputs with power-of-two lengths, the BISECTION algorithm of Nelson, Kieffer, and Cosman [16] gives essentially the same representation as a binary decision diagram [17]. Binary decision diagrams have been used widely in digital circuit analysis since the 1980s and also recently exploited for more general compression tasks [18], [19].

While these two lines of research led to the first clear articulation of the smallest grammar problem, its roots go back to much earlier work in the 1970s. In particular, Lempel and Ziv approached the problem from the direction of Kolmogorov complexity [20]. Over time, however, their work evolved toward data compression, beginning with a seminal paper [21] proposing the LZ77 compression algorithm. This procedure does *not* represent a string by a grammar. Nevertheless, we show in Section VII that LZ77 is deeply entwined with grammar-based compression. Lempel and Ziv soon produced another algorithm, LZ78, which did implicitly represent a string with a grammar [1]. We describe and analyze LZ78 in detail in Section VI. In 1984, Welch increased the efficiency of LZ78 with a new procedure, now known as LZW [2]. In practice, LZW is much preferred over LZ78, but for our purposes the difference is small.

Also in the 1970s, Storer and Szymanski explored a wide range of "macro-based" compression schemes [22]–[24]. They defined a collection of attributes that such a compressor might have, such as "recursive," "restricted," "overlapping," etc. Each combination of these adjectives described a different scheme, many of which they considered in detail and proved to be NP-hard.

Recently, the smallest grammar problem has received increasing interest in a broad range of communities. For example, de Marcken's thesis [9] investigated whether the structure of the smallest grammar generating a large, given body of English text could lead to insight about the structure of the language itself. Lanctot, Li, and Yang [8] proposed using the LONGEST MATCH algorithm for the smallest grammar problem to estimate the entropy of DNA sequences. Apostolico and Lonardi [11], [25], [10] suggested a scheme that we call GREEDY and applied it to the same problem. Larsson and Moffat proposed RE-PAIR [7] as a general, grammar-based algorithm. Most of these procedures are described and analyzed in Section VI.

There has also been an effort to develop algorithms that manipulate strings that are in compressed form. For example, Kida [14] and Shibata, *et al.* [26] have proposed pattern matching algorithms that run in time related not to the length of the searched string, but rather to the size of the grammar representing it. The relatively good performance of such algorithms represents a significant advantage of grammar-based compression over other compression techniques such as LZ77.

In short, the smallest grammar problem has been considered by many authors in many disciplines for many reasons over a span of decades. Given this level of interest, it is remarkable that the problem has not attracted greater attention in the general algorithms community.

III. Summary of Our Contributions

This paper makes four main contributions, enumerated below. Throughout, we use n to denote the length of an input string, and m^* to denote the size of the smallest grammar generating that same input string.

1) We show that the smallest grammar generating a given string is hard to approximate to within a small constant factor. Furthermore, we show that an $o(\log n/\log\log n)$ approximation would require progress on a well-studied problem in computational algebra.

Authorized licensed use limited to: Trial User - Warsaw University (Uniwersytet Warszawski). Downloaded on August 09,2023 at 11:58:59 UTC from IEEE Xplore. Restrictions apply.

2) We bound approximation ratios for several of the best known grammar-based compression algorithms. These results are summarized as follows:

Algorithm	Approximation Ratio Upper Bound	Lower Bound
LZ78	$O((n/\log n)^{\frac{2}{3}})$	$\Omega(n^{\frac{2}{3}}/\log n)$
BISECTION	$O((n/\log n)^{\frac{1}{2}})$	$\Omega(n^{\frac{1}{2}}/\log n)$
SEQUENTIAL	$O((n/\log n)^{\frac{3}{4}})$	$\Omega(n^{\frac{1}{3}})$
LONGEST MATCH	$O((n/\log n)^{\frac{2}{3}})$	$\Omega(\log\log n)$
GREEDY	$O((n/\log n)^{\frac{2}{3}})$	$> 1.37\ldots$
RE-PAIR	$O((n/\log n)^{\frac{2}{3}})$	$\Omega(\sqrt{\log n})$

The bounds for LZ78 hold for some variants, including LZW. Results for MPM mirror those for BISECTION. The lower bound for SEQUENTIAL extends to SEQUITUR.

3) We give new algorithms for the smallest grammar problem with exponentially better approximation ratios. First, we give a simple $O(\log^3 n)$ approximation. Then we provide a more complex $O(\log(n/m^*))$ approximation based on an entirely different approach.

4) We bring to light an intricate connection between grammar-based compression and the well-known LZ77 compression scheme.

The remainder of this paper is organized as follows. Section IV contains definitions and notational conventions, together with some basic lemmas. In Section V, we establish the hardness of the smallest grammar problem in two different and complementary senses. Then, in Section VI, we analyze the most widely known algorithms for the smallest grammar problem. Following this, we propose new algorithms in Section VII with approximation ratios that are exponentially better. Finally, Section VIII presents some of the many interesting lines of research radiating from this problem.

IV. PRELIMINARIES

This section introduces terminology, notation, and some basic lemmas about grammars that are used in later sections.

A. Grammars and Strings

A *grammar* G is a 4-tuple $(\Sigma, \Gamma, S, \Delta)$ in which 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. The last component of a grammar, denoted Δ, is a set of *rules* 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.

The *left side of a rule* $T \to \alpha$ is the symbol T, and the *right side of the rule* or *definition of* T is the string α. Similarly, the *left side of a grammar* consists of all nonterminals on the left sides of rules, and the *right side of a grammar* consists of all strings on the right sides of rules.

In the grammars we consider, there is exactly one rule $T \to \alpha$ in Δ for each nonterminal $T \in \Gamma$. Furthermore, all grammars are acyclic; that is, there exists an ordering of the nonterminals Γ such that each nonterminal precedes all the nonterminals in its definition. These properties guarantee that a grammar generates exactly one finite-length string.

A grammar naturally defines an *expansion* function of the form $(\Sigma \cup \Gamma)^* \mapsto \Sigma^*$. The expansion of a string is obtained by iteratively replacing each nonterminal by its definition until only terminals remain. We denote the expansion of a string α by $\langle\alpha\rangle$, and the length of the expansion of a string by $[\alpha]$; that is, $[\alpha] = |\langle\alpha\rangle|$. (In contrast, $|\alpha|$ denotes the length of the string α in the traditional sense; that is, the number of symbols in the string.) For the example grammar on the first page, we have $\langle A\rangle = \underline{\text{a rose}}$ and $[A] = 6$. The expansion of the start symbol $\langle S\rangle$ is the string *generated* by the grammar, and we typically refer to G as a grammar *for* the string $\langle S\rangle$.

The *size* of a grammar G is the total number of symbols in all definitions

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

We use several notational conventions to compactly express strings. The symbol $|$ represents a terminal that appears only once in a string. (For this reason, we refer to $|$ as a *unique symbol*.) When $|$ is used several times in the same string, each appearance represents a different symbol. For example, $a|bb|cc$ contains five distinct symbols and seven symbols in total.[1]

Product notation is used to express concatenation, and parentheses are used for grouping. For example

$$(ab)^5 = ababababab$$
$$\prod_{i=1}^{3} ab^i \mid = ab \mid abb \mid abbb \mid .$$

The input to the smallest grammar problem is never specified using such shorthand; we use it only for clarity of exposition in proofs, counterexamples, etc.

Finally, we observe the following variable-naming conventions throughout: terminals are lower case letters or digits, nonterminals are upper case letters, and strings of symbols are lower case Greek letters. In particular, σ denotes the input to a compression algorithm, and n denotes its length; that is, $n = |\sigma|$. The size of a particular grammar for σ is m, and the size of the smallest grammar is m^*. Unless otherwise stated, all logarithms are base two.

B. Approximation Ratio

Our focus is on the *approximation ratio* of algorithms for the smallest grammar problem. The approximation ratio of an algorithm A is a function $a(n)$ defined by

$$a(n) = \max_{x \in \Sigma^n} \left(\frac{\text{grammar size for } x \text{ produced by } A}{\text{size of the smallest grammar for } x} \right)$$

Thus, our focus is on the performance of algorithms in the worst case. The focus on worst case analysis is motivated by the goal

[1]For the lower bounds on LONGEST MATCH and RE-PAIR and in our hardness results, the use of unique symbols in the input implies that the alphabet size for these classes of examples grows unbounded. For the rest of the lower bounds, however, the alphabet sizes are fixed.

Authorized licensed use limited to: Trial User - Warsaw University (Uniwersytet Warszawski). Downloaded on August 09,2023 at 11:58:59 UTC from IEEE Xplore. Restrictions apply.

of establishing provable *guarantees* on performance, and therefore establishing a fair basis for comparing algorithms. In addition, the worst case analysis addresses an inherent problem with characterizing compression performance on low-entropy strings. Kosaraju and Manzini [27] point out that the standard notions of universality and redundancy are not meaningful measures of a compressor's performance on low-entropy strings. Our approximation ratio measure handles all cases and therefore sidesteps this issue.

C. Basic Lemmas

In this subsection, we give some easy lemmas that highlight basic points about the smallest grammar problem. In proofs here and elsewhere, we ignore the possibility of degeneracies where they raise no substantive issue, e.g., a nonterminal with an empty definition or a secondary nonterminal that never appears in a definition.

Lemma 1: The smallest grammar for a string of length n has size $\Omega(\log n)$.

Proof: Let G be an arbitrary grammar of size m. We show that G generates a string of length $O(3^{m/3})$, which implies the claim. Define a sequence of nonterminals recursively as follows. Let T_1 be the start symbol of grammar G. Let T_{i+1} be the nonterminal in the definition of T_i that has the longest expansion. (Break ties arbitrarily.) The sequence ends when a nonterminal T_n, defined only in terms of terminals, is reached. Note that the nonterminals in this sequence are distinct, since the grammar is acyclic.

Let k_i denote the length of the definition of T_i. Then the length of the expansion of T_i is upper-bounded by k_i times the length of the expansion of T_{i+1}. By an inductive argument, we find

$$[T_1] \leq k_1 \cdot k_2 \ldots k_n.$$

On the other hand, we know that the sum of the sizes of the definitions of $T_1 \ldots T_m$ is at most the size of the entire grammar

$$k_1 + k_2 + \cdots + k_n \leq m.$$

It is well known that a set of positive integers with sum at most m has product at most $3^{\lceil m/3 \rceil}$. Thus, the length of the string generated by G is $O(3^{m/3})$ as claimed. □

Next we show that certain highly structured strings are generated by small grammars.

Lemma 2: Let α be the string generated by grammar G_α, and let β be the string generated by grammar G_β. Then we have the following.

1) There exists a grammar of size $|G_\alpha| + |G_\beta| + 2$ that generates the string $\alpha\beta$.
2) There exists a grammar of size $|G_\alpha| + O(\log k)$ that generates the string α^k.

Proof: To establish 1), create a grammar containing all rules in G_α, all rules in G_β, and the start rule $S \rightarrow S_\alpha S_\beta$, where S_α is the start symbol of G_α and S_β is the start symbol of G_β.

For 2), begin with the grammar G_α, and call the start symbol A_1. We extend this grammar by defining nonterminals A_i with expansion α^i for various i. The start rule of the new grammar is A_k. If k is even (say, $k=2j$), define $A_k \rightarrow A_j A_j$ and define A_j recursively. If k is odd (say, $k=2j+1$), define $A_k \rightarrow A_j A_j A_1$ and again define A_j recursively. When $k = 1$, we are done. With each recursive call, the nonterminal subscript drops by a factor of at least two and at most three symbols are added to the grammar. Therefore, the total grammar size is $|G_\alpha| + O(\log k)$. □

Lemma 2 is helpful in lower-bounding the approximation ratios of certain algorithms when it is necessary to show that there exist small grammars for strings such as $a^{k(k+1)}\,(ba^k)^{(k+1)}$.

The following lemma is used extensively in our analysis of previously proposed algorithms. Roughly, it upper-bounds the complexity of a string generated by a small grammar.

Lemma 3 (mk Lemma): If a string σ is generated by a grammar of size m, then σ contains at most mk distinct substrings of length k.

Proof: Let G be a grammar for σ of size m. For each rule $T \rightarrow \alpha$ in G, we upper-bound the number of length-k substrings of $\langle T \rangle$ that are not substrings of the expansion of a nonterminal in α. Each such substring either begins at a terminal in α, or else begins with between 1 and $k-1$ terminals from the expansion of a nonterminal in α. Therefore, the number of such strings is at most $|\alpha| \cdot k$. Summing over all rules in the grammar gives the upper bound mk.

All that remains is to show that all substrings are accounted for in this calculation. To that end, let τ be an arbitrary length-k substring of σ. Find the rule $T \rightarrow \alpha$ such that τ is a substring of $\langle T \rangle$, and $\langle T \rangle$ is as short as possible. Thus, τ is a substring of $\langle T \rangle$ and is not a substring of the expansion of a nonterminal in α. Therefore, τ was indeed accounted for above. □

V. Hardness

We establish the hardness of the smallest grammar problem in two ways. First, we show that approximating the size of the smallest grammar to within a small constant factor is NP-hard. Second, we show that approximating the size to within $o(\log n / \log\log n)$ would require progress on an apparently difficult computational algebra problem. These two hardness arguments are curiously complementary, as we discuss in Section V-C.

A. NP-Hardness

Theorem 1: There is no polynomial-time algorithm for the smallest grammar problem with approximation ratio less than $8569/8568$ unless P = NP.

Proof: We use a reduction from a restricted form of vertex cover based closely on arguments by Storer and Szymanski [23], [24]. Let $H = (V, E)$ be a graph with maximum degree three and $|E| \geq |V|$. We can map the graph H to a string σ over an alphabet that includes a distinct terminal (denoted v_i) corresponding to each vertex $v_i \in V$ as follows:

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

There is a natural correspondence between vertex covers of the graph H and grammars for the string σ. In particular, we will show that the smallest grammar for σ has size $15|V| + 3|E| + k$,

Authorized licensed use limited to: Trial User - Warsaw University (Uniwersytet Warszawski). Downloaded on August 09,2023 at 11:58:59 UTC from IEEE Xplore. Restrictions apply.

where k is the size of the minimum vertex cover for H. However, the size of the minimum cover for this family of graphs is known to be hard to approximate below a ratio of $145/144$ unless P = NP [28]. Therefore, it is equally hard to approximate the size of the smallest grammar for σ below the ratio

$$\rho = \frac{15|V| + 3|E| + \frac{145}{144}k}{15|V| + 3|E| + k}.$$

Since all vertices in H have degree at most three, $|E| \leq \frac{3}{2}|V|$. Furthermore, since each vertex can cover at most three edges, the size of the minimum vertex cover, k, must exceed $\frac{1}{3}|E| \geq \frac{1}{3}|V|$. The expression above achieves its minimum when $|E|$ is large and k is small. From the constraints $|E| \leq \frac{3}{2}|V|$ and $k \geq \frac{1}{3}|V|$, we get the lower bound

$$\rho \geq \frac{15|V| + 3 \cdot \frac{3}{2}|V| + \frac{145}{144}\left(\frac{1}{3}|V|\right)}{15|V| + 3 \cdot \frac{3}{2}|V| + \left(\frac{1}{3}|V|\right)} = \frac{8569}{8568}.$$

Now we show that the minimal grammars for σ must assume a particular structure related to the vertex cover of H. Let G be an arbitrary grammar that generates σ. Suppose that there exists a nonterminal with an expansion of some form other than $\#v_i, v_i\#$, or $\#v_i\#$. Then that nonterminal either appears at most once in G or else expands to a single character, since no other substring of two or more characters appears multiple times in σ. Replacing each occurrence of this nonterminal by its definition and deleting its defining rule can only decrease the size of G. Thus, in searching for the smallest grammar for σ, we need only consider grammars in which every nonterminal has an expansion of the form $\#v_i, v_i\#$, or $\#v_i\#$.

Next, suppose grammar G does not contain a nonterminal with expansion $\#v_i$. Then this string must appear at least twice in the start rule, since the two occurrences generated by the first product term cannot be written another way. Adding a nonterminal with expansion $\#v_i$ costs two symbols, but also saves at least two symbols, and consequently gives a grammar no larger than G. Similar reasoning applies for strings of the form $v_i\#$. Thus, we need only consider grammars in which there are nonterminals with expansions $\#v_i$ and $v_i\#$ for all vertices v_i in the graph H.

Finally, let $C \subseteq V$ denote the set of vertices v_i such that G contains a rule for the substring $\#v_i\#$. Now suppose that C is not a vertex cover for H. Then there exists an edge $(v_i, v_j) \in E$ such that G does not contain rules for either $\#v_i\#$ or $\#v_j\#$. As a result, the occurrences of these strings generated by the second product term of σ must be represented by at least four symbols in the start rule of G. Furthermore, the string $\#v_i\#v_j\#$ generated by the third product term must be represented by at least three symbols. However, defining a nonterminal with expansion $\#v_i\#$ costs two symbols (since there is already a nonterminal with expansion $\#v_i$), but saves at least two symbols as well, giving a grammar no larger than before. Therefore, we need only consider grammars such that the corresponding set of vertices C is a vertex cover.

The size of a grammar with the structure described above is $8|V|$ for the first section of the start rule, plus $3|V| - |C|$ for the second section, plus $3|E|$ for the third section, plus $4|V|$ for rules for strings of the form $\#v_i$ and $v_i\#$, plus $2|C|$ for rules for strings of the form $\#v_i\#$, which gives $15|V| + 3|E| + |C|$. This quantity is minimized when C is a minimum vertex cover. In that case, the size of the grammar is $15|V| + 3|E| + k$ as claimed. □

B. Hardness Via Addition Chains

This subsection demonstrates the hardness of the smallest grammar problem in an alternative sense: a procedure with an approximation ratio $o(\log n/\log\log n)$ would imply progress on an apparently difficult algebraic problem in a well-studied area.

Consider the following problem. Let $k_1, k_2, \ldots, k_p$ be positive integers. How many multiplications are required to compute $x^{k_1}, x^{k_2}, \ldots, x^{k_p}$, where x is a real number? This problem has a convenient, alternative formulation. An *addition chain* is an increasing sequence of positive integers starting with 1 and with the property that every other term is the sum of two (not necessarily distinct) predecessors. The connection between addition chains and computing powers is straightforward: the terms in the chain indicate the powers to be computed. For example, $1, 2, 4, 8, 9, 18, 22, 23$ is an addition chain which computes x^9 and x^{23} using seven multiplications. The problem of computing, say, x^9 and x^{23} using the fewest multiplications is closely tied to the problem of finding the smallest grammar for the string $\sigma = x^9 \mid x^{23}$. Roughly speaking, a grammar for σ can be regarded as an algorithm for computing x^9 and x^{23} and *vice versa*. The following theorem makes these mappings precise.

Theorem 2: Let $T = \{k_1 \ldots k_p\}$ be a set of distinct positive integers, and define the string $\sigma = x^{k_1} \mid x^{k_2} \mid \ldots \mid x^{k_p}$. Let l^* be the length of the shortest addition chain containing T and let m^* be the size of the smallest grammar for the string σ. Then the following relationship holds:

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

Proof: We translate the grammar of size m^* for string σ into an addition chain containing T with length at most m^*. This will establish the left inequality $l^* \leq m^*$. For clarity, we accompany the description of the procedure with an example and some intuition. Let T be the set $\{9, 23\}$. Then $\sigma = x^9 \mid x^{23}$. The smallest grammar for this string has size $m^* = 13$

$$S \rightarrow A \mid AABxx$$
$$A \rightarrow BBB$$
$$B \rightarrow xxx.$$

We begin converting the grammar to an addition chain by ordering the rules so that their expansions increase in length. Then we underline symbols in the grammar according to the following two rules.

1) The first symbol in the first rule is underlined.
2) Every symbol preceded by a nonterminal or an x is underlined.

Thus, in the example, we would underline as follows:

$$B \rightarrow \underline{xxx}$$
$$A \rightarrow B\underline{BB}$$
$$S \rightarrow A \mid A\underline{ABxx}.$$

Each underlined symbol generates one term in the addition chain as follows. Starting from the underlined symbol, work leftward until the start of the definition or a unique symbol is

Authorized licensed use limited to: Trial User - Warsaw University (Uniwersytet Warszawski). Downloaded on August 09,2023 at 11:58:59 UTC from IEEE Xplore. Restrictions apply.

encountered. This span of symbols defines a substring which ends with the underlined symbol. The length of the expansion of this substring is a term in the addition chain. In the example, we would obtain the substrings

$$x, xx, xxx, BB, BBB, AA, AAB, AABx, AABxx$$

and the addition chain $1, 2, 3, 6, 9, 18, 21, 22, 23$.

Intuitively, the terms in the addition chain produced above are the lengths of the expansions of the secondary nonterminals in the grammar. But these alone do not quite suffice. To see why, note that the rule $T \to ABC$ implies that $[T] = [A]+[B]+[C]$. If we ensure that the addition chain contains $[A]$, $[B]$, and $[C]$, then we still cannot immediately add $[T]$ because $[T]$ is the sum of three preceding terms, instead of two. Thus, we must also include, say, the term $[AB]$, which is itself the sum of $[A]$ and $[B]$. The creation of such extra terms is what the elaborate underlining procedure accomplishes. With this in mind, it is easy to verify that the construction detailed above gives an addition chain of length at most m^* that contains T.

All that remains is to establish the second inequality $m^* \leq 4l^*$. We do this by translating an addition chain of length l into a grammar for the string σ of size at most $4l$. As before, we carry along an example. Let $T = \{9, 23\}$. The shortest addition chain containing T has length $l = 7$: $1, 2, 4, 5, 9, 18, 23$.

We associate the symbol x with the first term of the sequence and a distinct nonterminal with each subsequent term. Each nonterminal is defined using the symbols associated with two preceding terms, just as each term in the addition sequence is the sum of two predecessors. The start rule consists of the nonterminals corresponding to the terms in T, separated by uniques. In the example, this gives the following grammar:

$$\begin{aligned} T_2 &\to xx & T_4 &\to T_2T_2 \\ T_5 &\to T_4x & T_9 &\to T_5T_4 \\ T_{18} &\to T_9T_9 & T_{23} &\to T_{18}T_5 \\ & S \to T_9 \mid T_{23}. \end{aligned}$$

The start rule has length $2|T|-1 \leq 2l^*$, and the l^*-1 secondary rules each have exactly two symbols on the right. Thus, the total size of the grammar is at most $4l^*$. □

Addition chains have been studied extensively for decades (see surveys in Knuth [29] and Thurber [30]). In order to find the shortest addition chain containing a single, specified integer n, a subtle algorithm known as the *M-ary method* gives a $1 + O(1/\log\log n)$ approximation. (This is apparently folklore.) One writes n in a base M, which is a power of 2

$$n = d_0M^k + d_1M^{k-1} + d_2M^{k-2} + \cdots + d_{k-1}M + d_k.$$

The addition chain begins $1, 2, 3, \ldots, M-1$. Then one puts d_0, doubles it $\log M$ times, adds d_1 to the result, doubles that $\log M$ times, adds d_2 to the result, etc. The total length of the addition chain produced is at most

$$(M-1) + \log n + \frac{\log n}{\log M} = \log n + O(\log n/\log\log n).$$

In the expression on the left, the first term counts the first $M-1$ terms of the addition chain, the second counts the doublings, and the third counts the increments of d_i. The equality follows by choosing M to be the smallest power of two which is at least $\log n/\log\log n$.

The M-ary method is very nearly the best possible. Erdös [31] showed that, in a certain sense, the shortest addition chain containing n has length at least $\log n + \log n/\log\log n$ for almost all n. Even if exponentially more time is allowed, no exact algorithm (and apparently even no better approximation algorithm) is known.

The *general addition chain problem*, which consists of finding the shortest addition chain containing a specified set of integers $k_1 \ldots k_p$, is known to be NP-hard if the integers k_i are given in binary [32]. There is an easy $O(\log(\sum k_i))$ approximation algorithm. First, generate all powers of two less than or equal to the maximum of the input integers k_i. Then form each k_i independently by summing a subset of these powers corresponding to 1's in the binary representation of k_i. In 1976, Yao [33] pointed out that the second step could be tweaked in the spirit of the M-ary method. Specifically, he groups the bits of k_i into blocks of size $\log\log k_i - 2\log\log\log k_i$ and tackles all blocks with the same bit pattern at the same time. This improves the approximation ratio slightly to $O(\log n/\log\log n)$.

Yao's method retains a frustrating aspect of the naive algorithm: there is no attempt to exploit special relationships between the integers k_i; each one is treated independently. For example, suppose $k_i = 3^i$ for $i = 1$ to p. Then there exists a short addition chain containing all of the k_i: $\underline{1}, 2, \underline{3}, 6, \underline{9}, 18, \underline{27}, \ldots$. But Yao's algorithm effectively attempts to represent powers of three in base two.

However, even if the k_i are written in unary, apparently no polynomial time algorithm with a better approximation ratio than Yao's is known. Since Theorem 2 links addition chains and small grammars, finding an approximation algorithm for the smallest grammar problem with ratio $o(\log n/\log\log n)$ would require improving upon Yao's method.

C. An Observation on Hardness

We have demonstrated that the smallest grammar problem is hard to approximate through reductions from two different problems. Interestingly, there is also a marked difference in the types of strings involved.

Specifically, Theorem 1 maps graphs to strings with large alphabets and few repeated substrings. In such strings, the use of hierarchy does not seem to be much of an advantage. Thus, we show the NP-completeness of the smallest grammar problem by analyzing a class of input strings that specifically avoids the most interesting aspect of the problem: hierarchy.

On the other hand, Theorem 2 maps addition chain problems to strings over a unary alphabet (plus unique symbols). The potential for use of hierarchy in representing such strings is enormous; in fact, the whole challenge now is to construct an intricate hierarchy of rules, each defined in terms of the others. Thus, this reduction more effectively captures the most notable aspect of the smallest grammar problem.

Taken together, these two reductions show that the smallest grammar problem is hard in both a "combinatorial packing" sense and a seemingly orthogonal "hierarchical structuring" sense.

Authorized licensed use limited to: Trial User - Warsaw University (Uniwersytet Warszawski). Downloaded on August 09,2023 at 11:58:59 UTC from IEEE Xplore. Restrictions apply.

VI. Analysis of Previous Algorithms

In this section, we establish upper and lower bounds on the approximation ratios of six previously proposed algorithms for the smallest grammar problem: LZ78, Bisection, Sequential, Longest Match, Greedy, and Re-Pair. In addition, we discuss some closely related algorithms: LZW, Mpm, and Sequitur.

Although most of the algorithms in this section were originally designed as compression algorithms, we view them as approximation algorithms for the smallest grammar problem. Generally speaking, a good grammar-based compression algorithm should attempt to find the smallest possible grammar generating the input string. Nonetheless, there do exist disconnects between our theoretical study of the smallest grammar problem and practical data compression.

First, our optimization criteria is grammar size, whereas the optimization criteria in data compression is the bit length of the compressed string. A grammar with a smaller size does not necessarily translates into a smaller compression rate as described in [4]. However, a grammar of size m can be represented with at most $m \log m$ bits by assigning each distinct symbol a unique $(\log m)$-bit representation. Such a $\log m$ factor is small by the standards of our worst case theoretical analyses, but enormous by practical data compression standards.

Perhaps more importantly, data compression algorithms are typically designed with an eye toward *universality* (asymptotically optimal compression of strings generated by a finite-state source) and *low redundancy* (fast convergence to that optimum). Informally, strings generated by a finite-state source have high entropy; that is, they are compressible by only a constant factor. Thus, the main focus in the design of a data compressor is on high-entropy strings. In fact, Kosaraju and Manzini [27] point out that universality and redundancy are not meaningful measures of a compressor's performance on low-entropy strings. Consequently, performance on low-entropy strings is typically neglected completely.

The situation is quite different when one studies the worst case approximation ratio instead of universality and redundancy. If the smallest grammar for a high-entropy input string of length n has size, say, $n/\log n$, then any compressor can approximate the smallest grammar to within a $\log n$ factor. The low-entropy strings, however, present a serious challenge. If an input string is generated by a grammar of size, say, $n^{1/3}$, then a carelessly designed algorithm could exhibit an approximation ratio as bad as $n^{2/3}$. There is little hope that mapping the grammar to a binary string in a clever manner could offset such a failure. Thus, grammar-based data compressors and approximation algorithms can both be viewed as approaches to the smallest grammar problem, but they target different ranges of inputs.

Finally, practical data compression mandates linear running time in the length of the input string, with particular attention to the specific constants hidden by asymptotic notation. Ideally, a compressor should also be on-line; that is, a single left-to-right pass through the input string should suffice. Space consumption throughout this pass should, preferably, be a function of the size of the compressed string, not the size of the string being compressed.

As a result of these disconnects, one must take the results in the remainder of this section with a caveat: while we show that many grammar-based data compression algorithms exhibit mediocre approximation ratios, the designers of these algorithms were concerned with slightly different measures, different inputs, and many practical issues that we ignore.

A. LZ78

The well-known LZ78 compression scheme was proposed by Lempel and Ziv [1]. In traditional terms, the LZ78 algorithm represents a string σ by a sequence of pairs. Each pair represents a substring of σ, and is of the form (i, c), where i is an integer and c is a symbol in σ. If i is zero, then the expansion of the pair is simply c. Otherwise, the expansion is equal to the expansion of the ith pair followed by the symbol c. The concatenation of the expansions of all pairs is σ. For example, the following sequence:

$$(0, a)\ (1, b)\ (0, b)\ (2, a)\ (3, a)\ (2, b)$$

represents the string $a\,ab\,b\,aba\,ba\,abb$, where spaces are added to clarify the correspondence.

The sequence-of-pairs representation of a string is generated by LZ78 in a single left-to-right pass as follows. Begin with an empty sequence of pairs. At each step, while there is input to process, find the shortest, nonempty prefix of the remaining input that is not the expansion of a pair already in the sequence. There are two cases.

1) If this prefix consists of a single symbol c, then append the pair $(0, c)$ to the sequence.
2) Otherwise, this prefix must be of the form αc, where α is the expansion of some pair already in the sequence (say, the ith one) and c is a symbol. In this case, append the pair (i, c) to the sequence.

For a cleaner analysis, we assume that an implicit "end-of-file" character is appended to the string in order to guarantee that one of the above two cases always applies. This special character is omitted from the examples below for clarity.

1) LZ78 in Grammar Terms: An LZ78 pair sequence maps naturally to a grammar. Associate a nonterminal T with each pair (i, c). If i is zero, define the nonterminal by $T \to c$. Otherwise, define the nonterminal to be $T \to Uc$, where U is the nonterminal associated with the ith pair. The right side of the start rule contains all the nonterminals associated with pairs. For example, the grammar associated with the example sequence is as follows:

$$\begin{array}{lll} S \to X_1X_2X_3X_4X_5X_6 & & \\ X_1 \to a & X_3 \to b & X_5 \to X_3a \\ X_2 \to X_1b & X_4 \to X_2a & X_6 \to X_2b \end{array}$$

Given this easy mapping, hereafter we simply regard the output of LZ78 as a grammar rather than as a sequence of pairs.

Note that the grammars produced by LZ78 are of a restricted form in which the right side of each secondary rule contains at most two symbols and at most one nonterminal. Subject to these restrictions, the smallest grammar for even the string x^n has size

Authorized licensed use limited to: Trial User - Warsaw University (Uniwersytet Warszawski). Downloaded on August 09,2023 at 11:58:59 UTC from IEEE Xplore. Restrictions apply.

$\Omega(\sqrt{n})$. (On the other hand, grammars with such a regular form can be more efficiently encoded into bits.)

The next two theorems provide nearly matching upper and lower bounds on the approximation ratio of LZ78 when it is interpreted as an approximation algorithm for the smallest grammar problem.

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

Proof: The lower bound follows by analyzing the behavior of LZ78 on input strings of the form

$$\sigma_k = a^{k(k+1)/2}(ba^k)^{(k+1)^2}$$

where $k > 0$. The length of this string is $n = \Theta(k^3)$. Repeated application of Lemma 2 implies that there exists a grammar for σ_k of size $O(\log k) = O(\log n)$.

The string σ_k is processed by LZ78 in two stages. During the first, the $k(k+1)/2$ leading a's are consumed and nonterminals with expansions $a, aa, aaa, \ldots, a^k$ are created. During the second stage, the remainder of the string is consumed and a nonterminal with expansion $a^i b a^j$ is created for all i and j between 0 and k. For example, σ_4 is represented by nonterminals with expansions as indicated as follows:

a	aa	aaa	$aaaa$
b	$aaaab\ aaaaba$	$aaab\ aaaabaa$	$aab\ aaaabaaa$
	$ab\ aaaabaaaa$		
ba	$aaaba\ aaabaa$	$aaba\ aaabaaa$	$aba\ aaabaaaa$
baa	$aabaa\ aabaaa$	$abaa\ aabaaaa$	
$baaa$	$abaaa\ abaaaa$		
$baaaa$			

The pattern illustrated above can be shown to occur in general with an induction argument. As a result, the grammar produced by LZ78 has size $\Omega(k^2) = \Omega(n^{2/3})$. Dividing by our upper bound on the size of the smallest grammar proves the claim. □

Theorem 4: The approximation ratio of LZ78 is $O((n/\log n)^{2/3})$.

Our techniques in the following Proof of Theorem 4 form the basis for two other upper bounds presented in this section. The core idea is that nonterminals must expand to distinct substrings of the input. By the mk Lemma, however, there are very few short distinct substrings of the input. Thus most nonterminals expand to long substrings. However, the total expansion length of all nonterminals must be equal to the size of the input. As a result, there cannot be too many nonterminals in the grammar.

Proof: Suppose that the input to LZ78 is a string σ of length n, and that the smallest grammar generating σ has size m^*. Let $S \to X_1 \ldots X_p$ be the start rule generated by LZ78. First observe that the size of the LZ78 grammar is at most $3p$, since each nonterminal X_i is used once in the start rule and is defined using at most two symbols. Therefore, it suffices to upper-bound p, the number of nonterminals in the start rule.

To that end, list the nonterminals of the grammar in order of increasing expansion length. Group the first m^* of these nonterminals, the next $2m^*$, the next $3m^*$, and so forth. Let g be the number of complete groups of nonterminals that can be formed in this way. By this definition of g, we have

$$m^* + 2m^* + \cdots + gm^* + (g+1)m^* > p$$

and so $p = O(g^2 m^*)$.

On the other hand, the definition of LZ78 guarantees that each nonterminal X_i expands to a distinct substring of σ. Moreover, Lemma 3 states that σ contains at most m^*k distinct substrings of length k. Thus, there can be at most m^* nonterminals which have expansion length 1, and at most $2m^*$ nonterminals which have expansion length 2, and so on.

It follows that each nonterminal in the jth group must expand to a string of length at least j. Therefore, we have

$$\begin{aligned} n &= [X_1] + \cdots + [X_p] \\ &\geq 1^2 m^* + 2^2 m^* + 3^2 m^* + \cdots + g^2 m^* \end{aligned}$$

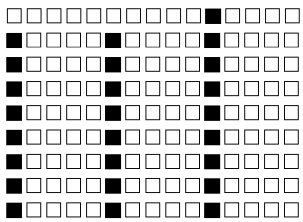
and so $g = O((n/m^*)^{1/3})$. The inequality follows since we are ignoring the incomplete $(g+1)$th group.

Substituting this bound on g into the upper bound on p obtained previously gives

$$p = O\left(\left(\frac{n}{m^*}\right)^{2/3} m^*\right) = O\left(\left(\frac{n}{\log n}\right)^{2/3} m^*\right)$$

The second equality follows from Lemma 1, which says that the smallest grammar for a string of length n has size $\Omega(\log n)$. □

2) LZW: Some practical improvements on LZ78 are embodied in a later algorithm, LZW [2]. The grammars implicitly generated by the two procedures are not substantively different, but LZW is more widely used in practice. For example, it is used to encode images in the popular `gif` format. Interestingly, the bad strings introduced in Theorem 3 have a natural graphical interpretation. Below, σ_4 is written in a 15×9 grid pattern using □ and ■ for a and b, respectively.



Thus, an image with colors in this simple vertical stripe pattern yields a worst case string in terms of approximation ratio. This effect can be observed in practice on even small examples. For example, a 68×68 image consisting of four horizontal lines spaced 16 pixels apart is stored by Corel PhotoPaint, a commercial graphics program, in a 933-yte file. When the image is rotated 90° to create vertical lines instead, the stored file grows to 1142 byte.

B. BISECTION

The BISECTION algorithm was proposed by Kieffer, Yang, Nelson, and Cosman [3], [16]. For binary input strings of length 2^k, the same technique was employed much earlier in binary decision diagrams, a data structure used to represent and easily manipulate Boolean functions.

Authorized licensed use limited to: Trial User - Warsaw University (Uniwersytet Warszawski). Downloaded on August 09,2023 at 11:58:59 UTC from IEEE Xplore. Restrictions apply.

1) The Procedure: BISECTION works on an input string σ as follows. Select the largest integer j such that $2^j < |\sigma|$. Partition σ into two substrings with lengths 2^j and $|\sigma| - 2^j$. Repeat this partitioning process recursively on each substring produced that has length greater than one. Afterward, 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.

Example 1. Consider the string $\sigma = 1110111010011$. We recursively partition and associate a nonterminal with each distinct substring generated as shown as follows:

$$\underbrace{1110111010011}_{S} \qquad S \to T_1T_2$$

$$\underbrace{11101110}_{T_1}\ \underbrace{10011}_{T_2} \qquad T_1 \to U_1U_1\ T_2 \to U_21$$

$$\underbrace{1110}_{U_1}\ 1110\ \underbrace{1001}_{U_2}\ 1 \qquad U_1 \to V_1V_2\ U_2 \to V_2V_3$$

$$\underbrace{11}_{V_1}\ \underbrace{10}_{V_2}\ 111010\ \underbrace{01}_{V_3}\ 1 \quad V_1 \to 11\ V_2 \to 10\ V_3 \to 01.$$

2) Bounds: The following two theorems give nearly matching lower and upper bounds on the approximation ratio of BISECTION.

Theorem 5: The approximation ratio of BISECTION is $\Omega(\sqrt{n}/\log n)$.

Proof: We analyze the behavior of BISECTION on input strings of the form

$$\sigma_k = a(b^{2^k}a)^{2^k-1}$$

where $k > 0$. This string has length $n = 2^{2k}$. After k bisections, σ_k is partitioned into 2^k distinct substrings of length 2^k. In particular, each contains a single a, which appears in the ith position in the ith substring. For example, bisecting σ_2 twice gives four distinct strings: *abbb babb bbab bbba*.

A routine induction argument shows that this pattern holds in general for σ_k. Since each distinct substring generates a nonterminal, BISECTION produces a grammar of size $\Omega(2^k) = \Omega(\sqrt{n})$ on input σ_k.

On the other hand, Lemma 2 implies that there exists a grammar for σ_k of size $O(k) = O(\log n)$. The approximation ratio of $\Omega(\sqrt{n}/\log n)$ follows. □

Theorem 6: The approximation ratio of BISECTION is $O(\sqrt{n/\log n})$.

Proof: Suppose that the input to BISECTION is a string σ of length n, and that the smallest grammar-generating σ has size m^*. Let j be the largest integer such that $2^j \le n$. Note that the size of the BISECTION grammar for σ is at most twice the number of distinct substrings generated during the recursive partitioning process. Thus, it suffices to upper-bound the latter quantity.

At most one string at each level of the recursion has a length that is not a power of two; therefore, there are at most j strings with irregular lengths. All remaining strings have length 2^i for some i between 1 and j. We can upper-bound the number of these remaining strings in two ways. On one hand, BISECTION creates at most one string of length 2^j, at most two of length 2^{j-1}, at most four of length 2^{j-2}, etc. On the other hand, Lemma 3 says that σ contains at most $2^i m^*$ distinct substrings of length 2^i. The first observation gives a good upper bound on the number of distinct long strings generated by the recursive partitioning process, and the second is tighter for short strings. Putting this all together, the size of the BISECTION grammar is at most

$$\begin{aligned} 2 \cdot &\left(j + \sum_{i=1}^{\frac{1}{2}(j-\log j)} m^* 2^i + \sum_{i=\frac{1}{2}(j-\log j)}^{j} 2^{j-i} \right) \\ &= O(\log n) + O\left(m^* \sqrt{\frac{n}{\log n}} \right) + O(\sqrt{n \log n}) \\ &= O\left(m^* \sqrt{\frac{n}{\log n}} \right) \end{aligned}$$

In the second equation, we use the fact that $m^* = \Omega(\log n)$ by Lemma 1. □

3) MPM: BISECTION was generalized to an algorithm called MPM [3], which permits a string to be split more than two ways during the recursive partitioning process and allows that process to terminate early. For reasonable parameters, performance bounds are the same as for BISECTION.

C. SEQUENTIAL

Nevill–Manning and Witten introduced the SEQUITUR algorithm [6], [13]. Kieffer and Yang subsequently offered a similar, but improved algorithm that we refer to here as SEQUENTIAL [4].

1) The Procedure: SEQUENTIAL processes a string 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. If no prefix matches the expansion of a secondary nonterminal, then append the first terminal in the unprocessed portion of the input to the start rule. In either case, if the newly created pair of symbols at the end of the start rule already appears elsewhere in the grammar without overlap, then replace both occurrences by a new nonterminal whose definition is that pair. Finally, if some nonterminal occurs only once after this substitution, replace it by its definition, and delete the corresponding rule.

Example 2. As an example, consider the input string $\sigma = xxxxx\diamondsuit xxxxx\heartsuit$. After three steps, the grammar is: $S \to xxx$. When the next x is appended to the start rule, there are two copies of the substring xx. Therefore, a new rule, $R_1 \to xx$, is added to the grammar and both occurrences of xx are replaced by R_1 to produce the following intermediate grammar:

$$S \to R_1R_1$$
$$R_1 \to xx.$$

During the next two steps, the start rule expands to $S \to R_1R_1x\diamondsuit$. At this point, the expansion of R_1 is a prefix of the unprocessed part of σ, so the next two steps consume xx and append R_1 to S twice.

Authorized licensed use limited to: Trial User - Warsaw University (Uniwersytet Warszawski). Downloaded on August 09,2023 at 11:58:59 UTC from IEEE Xplore. Restrictions apply.

Now the pair R_1R_1 appears twice in S, and so a new rule $R_2 \rightarrow R_1R_1$ is added and applied

$$\begin{aligned} S &\rightarrow R_2x\diamondsuit R_2 \\ R_1 &\rightarrow xx \\ R_2 &\rightarrow R_1R_1. \end{aligned}$$

In the next step, x is consumed and now R_2x appears twice. A new rule $R_3 \rightarrow R_2x$ is created and substituted into S. Notice that the rule R_2 only appears once after this substitution. Therefore, the occurrence of R_2 in the definition of R_3 is replaced with R_1R_1, and R_2 is removed from the grammar. After the next step, we have the following final output:

$$\begin{aligned} S &\rightarrow R_3\diamondsuit R_3\heartsuit \\ R_1 &\rightarrow xx \\ R_3 &\rightarrow R_1R_1x. \end{aligned}$$

2) Bounds: The next two theorems bound the approximation ratio of SEQUENTIAL. Both the upper and lower bounds are considerably more complex than the analysis for LZ78 and BISECTION.

Theorem 7: The approximation ratio of SEQUENTIAL is $\Omega(n^{1/3})$.

Proof: We analyze the behavior of SEQUENTIAL on strings σ_k for $k > 0$, defined below, over an alphabet consisting of four symbols: $\{x, y, \diamondsuit, \heartsuit\}$

$$\begin{aligned} \sigma_k &= \alpha\diamondsuit\beta^{k/2} \\ \alpha &= x^{k+1}\diamondsuit x^{k+1}\heartsuit\delta_0\diamondsuit\delta_0\heartsuit\delta_1\diamondsuit\delta_1\heartsuit\ldots\delta_k\diamondsuit\delta_k\heartsuit \\ \beta &= \delta_k\delta_k\ \delta_k\delta_{k-1}\ \delta_k\delta_{k-2}\ \delta_k\delta_{k-3}\ \ldots\ \delta_k\delta_{k/2}\,x^k \\ \delta_i &= x^iyx^{k-i}. \end{aligned}$$

As SEQUENTIAL processes the prefix α, it creates nonterminals for the strings x^{2^i} for each i from 1 to $\log(k+1)$, a nonterminal with expansion x^{k+1}, a nonterminal with expansion δ_i for each i from 0 to k, and some nonterminals with shorter expansions that are not relevant here. With regard to the third assertion, note that SEQUENTIAL parses the first occurrence of the string δ_i in some particular way. It then consumes the $\diamondsuit$, and proceeds to consume the second occurrence of δ_i in exactly the same way as the first one. This process generates a nonterminal with expansion δ_i. Notice that the $\diamondsuit$ and $\heartsuit$ symbols are never added to a secondary rule.

The remainder of the input, the string $\beta^{k/2}$, is consumed in segments of length $k+1$. This is because, at each step, the leading $k+1$ symbols of the unprocessed portion of the input string are of the form x^{k+1} or δ_i for some i. Consequently, the corresponding nonterminal is appended to the start rule at each step.

At a high level, this is the inefficiency that we exploit. The length of β is *not* a multiple of $k+1$. As a result, each copy of β is represented by a different sequence of nonterminals.

Now we describe the parsing of $\beta^{k/2}$ in more detail. The first copy of β is parsed almost as it is written above. The only difference is that the final x^k at the end of this first copy is combined with the leading zero in the second copy of β and read as a single nonterminal. Thus, nonterminals with the following expansions are appended to the start rule as the first copy of β is processed:

$$\delta_k\delta_k\ \delta_k\delta_{k-1}\ \delta_k\delta_{k-2}\ \delta_k\delta_{k-3}\ \ldots\ \delta_k\delta_{k/2}\,x^{k+1}.$$

SEQUENTIAL parses the second copy of β differently, since the leading zero of this second copy has already been processed. Furthermore, the final x^{k-1} in the second copy of β is combined with the two leading zeroes in the third copy and read as a single nonterminal

$$\delta_{k-1}\delta_{k-1}\ \delta_{k-1}\delta_{k-2}\ \delta_{k-1}\delta_{k-3}\ \ldots\ \delta_{k-1}\delta_{k/2-1}\,x^{k+1}.$$

With two leading zeros already processed, the third copy of β is parsed yet another way. In general, an induction argument shows that the jth copy (indexed from 0) is read as

$$\delta_{k-j}\delta_{k-j}\ \delta_{k-j}\delta_{k-j-1}\ \delta_{k-j}\delta_{k-j-2}\ \ldots\ \delta_{k-j}\delta_{k/2-j}\,x^{k+1}.$$

No consecutive pair of nonterminals ever appears twice in this entire process, and so no new rules are created. Since the input string contains $k/2$ copies of β and each is represented by about k nonterminals, the grammar generated by SEQUENTIAL has size $\Omega(k^2)$.

On the other hand, there exists a grammar for σ_k of size $O(k)$. First, create a nonterminal X_i with expansion x^i for each i up to $k+1$. Each such nonterminal can be defined in terms of its predecessors using only two symbols: $X_i \rightarrow xX_{i-1}$. Next, define a nonterminal D_i with expansion δ_i for each i using three symbols: $D_i \rightarrow X_iyX_{k-i}$. Now it is straightforward to define nonterminals A and B which expand to α and β, respectively. Finally, using Lemma 2, $O(\log k)$ additional symbols suffice to define a start symbol with expansion $\alpha\diamondsuit\beta^{k/2}$. In total, this grammar has size $O(k)$. Therefore, the approximation ratio of SEQUENTIAL is $\Omega(k) = \Omega(n^{1/3})$. □

3) Irreducible Grammars: Our upper bound on the approximation ratio of SEQUENTIAL relies on a property of the output. In particular, Kieffer and Yang [4] show that SEQUENTIAL produces an *irreducible* grammar; that is, one which has the following three properties.

- (I1) All nonoverlapping pairs of adjacent symbols of the grammar are distinct.
- (I2) Every secondary nonterminal appears at least twice on the right side of the grammar.
- (I3) No two nonterminals in the grammar have the same expansion.

In upper-bounding the approximation ratio of SEQUENTIAL, we rely on properties of irreducible grammars established in the following two lemmas.

Lemma 4: The sum of the lengths of the expansions of all distinct nonterminals in an irreducible grammar is at most $2n$.

(This result also appears in [5, Appendix B, eq. (9.33)].)

Proof: Let S be the start symbol of an irreducible grammar for a string of length n, and let $R_1 \ldots R_k$ be the secondary nonterminals. Observe that the sum of the expansion lengths of all symbols on the left side of the grammar must be equal to the sum of the expansion lengths of all symbols on the right side of the grammar. Furthermore, every secondary

Authorized licensed use limited to: Trial User - Warsaw University (Uniwersytet Warszawski). Downloaded on August 09,2023 at 11:58:59 UTC from IEEE Xplore. Restrictions apply.

nonterminal appears at least twice on the right side by (I2). Therefore, we have

$$[S] + [R_1] + \cdots + [R_k] \geq 2([R_1] + \cdots + [R_k]).$$

Adding $[S] - [R_1] + \cdots + [R_k]$ to both side of this inequality and noting that $2[S] = 2n$ finishes the proof. □

Lemma 5: Every irreducible grammar of size m contains at least $2m/9$ distinct, nonoverlapping pairs of adjacent symbols.

Proof: For each rule, group the first and second symbols on the right side to form one pair, the third and fourth for a second pair, and so forth. If a rule has an odd number of symbols, ignore the last one.

We must show that only a few symbols are ignored in this process. In particular, we ignore at most $m/3$ *lonely* symbols, which appear alone on the right side of a rule. Each such rule accounts for three distinct symbols in the grammar: one in the rule's definition and at least two for the occurrences of the nonterminal defined by the rule. Thus, there can be at most $m/3$ such *lonely* symbols.

Among rules with two or more symbols on the right, at least $\frac{2}{3}$ of those symbols are in pairs. (The worst case is a rule of length 3.) Thus at least $\frac{2}{3}(m - (m/3)) = \frac{4m}{9}$ symbols must have been paired in our process. Putting this all together, there are at least $\frac{2m}{9}$ nonoverlapping pairs of adjacent symbols. □

4) Upper Bound: We can now upper-bound the approximation ratio of SEQUENTIAL by using the fact that SEQUENTIAL always produces an irreducible grammar.

Theorem 8: Every irreducible grammar for a string is $O((n/\log n)^{3/4})$ times larger than the size of the smallest grammar for that string.

Corollary 1: The approximation ratio of SEQUENTIAL is $O((n/\log n)^{3/4})$.

Proof (of Theorem 8): This argument closely follows the one used in Theorem 4. As before, let m be the size of an irreducible grammar generating a string σ of length n, and let m^* be the size of the smallest grammar.

Identify $2m/9$ distinct, nonoverlapping pairs of adjacent symbols in the irreducible grammar that are guaranteed to exist by Lemma 5. Note that at most $k-1$ pairs can expand to the same length-k substring of σ. To see why, suppose there are k or more pairs that expand to represent the same length-k substring of σ. The first nonterminal in each such pair must expand to a string with length between 1 and $k-1$. Hence, by pigeonholing, there must exist two pairs UV and XY such that $\langle U \rangle = \langle X \rangle$ and $\langle V \rangle = \langle Y \rangle$. Since all pairs are distinct, either $U \neq X$ or $V \neq Y$. In either case, we have two distinct symbols with the same expansion, which violates (I3).

List the pairs in order of increasing expansion length and group the first $1 \cdot 2m^*$ of these pairs, the next $2 \cdot 3m^*$, etc. Let g be the number of *complete* groups formed in this way. Then we have

$$1 \cdot 2m^* + 2 \cdot 3m^* + \cdots + g \cdot (g+1)m^* + (g+1) \cdot (g+2)m^* > \frac{2m}{9}.$$

And so $m = O(g^3 m^*)$. Lemma 3 implies that σ contains at most $m^* k$ distinct substrings of length k. As in Theorem 4, at most $m^* k(k-1)$ pairs have an expansion of length k. Consequently, each pair in the jth group expands to a string of length at least $j+1$. Thus, the total length of the expansions of all pairs is at least $1 \cdot 2^2 m^* + 2 \cdot 3^2 m^* + \ldots + g(g+1)^2 m^*$.

The $2m/9$ pairs constitute a subset of the symbols on the right side of the grammar. The total expansion length of all symbols on the right side of the grammar is equal to the total expansion length of all symbols on the left. Lemma 4 upper-bounds the latter quantity by $2n$. Therefore, we have

$$1 \cdot 2^2 m^* + 2 \cdot 3^2 m^* + \cdots + g(g+1)^2 m^* \leq 2n.$$

As a result, $g = O((n/m^*)^{1/4})$. As before, substituting this upper bound on g into the upper bound on m obtained previously implies the theorem

$$m = O((n/m^*)^{3/4} m^*) = O((n/\log n)^{3/4} m^*). \qquad \square$$

D. Global Algorithms

The remaining algorithms analyzed in this section all belong to a single class, which we refer to as *global algorithms*. We upper-bound the approximation ratio of every global algorithm by $O((n/\log n)^{2/3})$ with a single theorem. However, our lower bounds are all different, complex, and weak. Moreover, the lower bounds rely on strings over unbounded alphabets. Thus, it may be that every global algorithm has an excellent approximation ratio. Because they are so natural and our understanding is so incomplete, global algorithms are one of the most interesting topics related to the smallest grammar problem that deserve further investigation.

1) The Procedure: A global algorithm begins with the grammar $S \rightarrow \sigma$. The remaining work is divided into rounds. During each round, one selects a *maximal string* γ. (Global algorithms differ only in the way they select a maximal string in each round.) A maximal string γ has three properties.

- (M1) It has length at least two.
- (M2) It appears at least twice on the right side of the grammar without overlap.
- (M3) No strictly longer string appears at least as many times on the right side without overlap.

After a maximal string γ is selected, a new rule $T \rightarrow \gamma$ is added to the grammar. This rule is then applied by working left-to-right through the right side of every other rule, replacing each occurrence of γ by the symbol T. The algorithm terminates when no more maximal strings exist.

Example 3. An example illustrates the range of moves available to a global algorithm. (Throughout this section, we will use the input string $\sigma = abcabcabcabcaba$ for our examples.) We initially create the grammar $S \rightarrow abc\ abc\ abc\ abc\ ab\ a$ where spaces are added for clarity. The maximal strings are ab, abc, and $abcabc$. Suppose that we select the maximal string ab, and introduce the rule $T \rightarrow ab$. The grammar becomes

$$S \rightarrow Tc\,Tc\,Tc\,Tc\,T\,a$$
$$T \rightarrow ab$$

Now the maximal strings are Tc and $TcTc$. Suppose that we select $TcTc$. Then we add the rule $U \rightarrow TcTc$, and the definition of S becomes $S \rightarrow U\,U\,T\,a$. Now the only

Authorized licensed use limited to: Trial User - Warsaw University (Uniwersytet Warszawski). Downloaded on August 09,2023 at 11:58:59 UTC from IEEE Xplore. Restrictions apply.

maximal string is Tc. Adding the rule $V \rightarrow Tc$ yields the final grammar

$$S \rightarrow U\ U\ T\ a$$
$$T \rightarrow ab$$
$$U \rightarrow V\ V$$
$$V \rightarrow Tc.$$

2) Upper Bound: The approximation ratio of every global algorithm is $O((n/\log n)^{2/3})$. This follows from the fact that grammars produced by global algorithms are particularly well conditioned; not only are they irreducible, but they also possess an additional property described in the following lemma.

Lemma 6: Every grammar produced by a global algorithm has the following property. Let α and β be strings of length at least two on the right side. If $\langle\alpha\rangle = \langle\beta\rangle$, then $\alpha = \beta$.

Proof: We show that this is actually an invariant property of the grammar maintained throughout the execution of a global algorithm. The invariant holds trivially for the initial grammar $S \rightarrow \sigma$. So suppose that the invariant holds for grammar G, and then grammar G' is generated from G by introducing a new rule $T \rightarrow \gamma$. Let α' and β' be strings of length at least two on the right side of G' such that $\langle\alpha'\rangle = \langle\beta'\rangle$. We must show that $\alpha' = \beta'$.

There are two cases to consider. First, suppose that neither α' nor β' appear in γ. Then α' and β' must be obtained from nonoverlapping strings α and β in G such that $\langle\alpha\rangle = \langle\alpha'\rangle$ and $\langle\beta\rangle = \langle\beta'\rangle$. Since the invariant holds for G, we have $\alpha = \beta$. But then α and β are transformed the same way when the rule $T \rightarrow \gamma$ is added; that is, corresponding instances of the string γ within α and β are replaced by the nonterminal T. Therefore, $\alpha' = \beta'$. Otherwise, suppose that at least one of α' or β' appears in γ. Then neither α' nor β' can contain T. Therefore, both α' and β' appear in grammar G, where the invariant holds, and so $\alpha' = \beta'$ again. □

Lemma 7: Every grammar produced by a global algorithm is irreducible.

Proof: We must show that a grammar produced by a global algorithm satisfies the three properties of an irreducible grammar.

Propoert (I1). First, note that all nonoverlapping pairs of adjacent symbols on the right side are distinct since a global algorithm does not terminate until this condition holds.

Property (I2). We must show that every secondary nonterminal appears at least twice on the right side of the grammar. This property is also an invariant maintained during the execution of a global algorithm.

The property holds vacuously for the initial grammar $S \rightarrow \sigma$. Suppose that the property holds for a grammar G which has been generated by a global algorithm, and then we obtain a new grammar G' by introducing a new rule $T \rightarrow \gamma$ where γ is a maximal string. By the definition of maximal string, the nonterminal T must appear at least twice on the right side of G'. If γ contains only terminals or nonterminals which appear twice on the right side of G', then the invariant clearly holds for G'. Suppose, by contradiction, that γ contains a nonterminal V which appears only once on the right side of G'. Let $V \rightarrow \beta$ be the definition of V in G'. This implies that V only appears in the definition of T, and therefore the string β occurs exactly as many times as γ in G. Since γ is maximal, it must have length at least two, and therefore $[\gamma] > [\beta]$. In particular, this implies that during the step in which the rule for V was introduced, the intermediate grammar at that point contained a strictly longer string which appeared exactly the same number of times, which contradicts the assumption that G has been produced by a global algorithm.

Property (I3). Finally, we must show that distinct symbols have distinct expansions, unless the start symbol expands to a terminal. Once again, we use an invariant argument. The following invariants hold for every secondary rule $U \rightarrow \gamma$ in the grammar maintained during the execution of a global algorithm.

1) The string γ appears nowhere else in the grammar.
2) The length of γ is at least two.

Both invariants hold trivially for the initial grammar $S \rightarrow \sigma$. Suppose that the invariants hold for every rule in a grammar G, and then we obtain a new grammar G' by introducing the rule $T \rightarrow \delta$.

First, we check that the invariants hold for the new rule. The string δ cannot appear elsewhere in the grammar; such an instance would have been replaced by the nonterminal T. Furthermore, the length of δ is at least two, since δ is a maximal string.

Next, we check that the invariant holds for each rule $U \rightarrow \gamma'$ in G' that corresponds to a rule $U \rightarrow \gamma$ in G. If γ' does not contain T, then both invariants carry over from G. Suppose that γ' does contain T. The first invariant still carries over from G. The second invariant holds unless $\delta = \gamma$. However, since δ is a maximal string, that would imply that γ appeared at least twice in G, violating the first invariant.

The third property of an irreducible grammar follows from these two invariants. No secondary nonterminal can expand to a terminal, because the second invariant implies that each secondary nonterminal has an expansion of length at least two. No two nonterminals can expand to the same string either; their definitions have length at least two by the second invariant, and therefore their expansions are distinct by Lemma 6. □

Theorem 9: The approximation ratio of every global algorithm is $O((n/\log n)^{2/3})$.

Proof: This argument is similar to the upper bound on irreducible grammars and LZ78. Suppose that on input σ of length n, a global algorithm outputs a grammar G of size m, but the smallest grammar has size m^*. First note that G is irreducible by Lemma 7.

As before, list $2m/9$ distinct, nonoverlapping pairs of adjacent symbols in G (guaranteed to exist by Lemma 5) in order of increasing expansion length. This time, group the first $2m^*$ pairs, the next $3m^*$, and so forth, so that g complete groups can be formed. Therefore, we have

$$2m^* + 3m^* + \cdots + (g+1)m^* > 2m/9$$

which implies $m = O(g^2 m^*)$.

Lemma 6 implies that every pair expands to a distinct substring of σ. With Lemma 3, this implies every pair in the jth group expands to a string of length at least $j+1$. As before, the total length of the expansions of all pairs must be at least

$$2^2 m^* + 3^2 m^* + \cdots + g^2 m^* \leq 2n.$$

Authorized licensed use limited to: Trial User - Warsaw University (Uniwersytet Warszawski). Downloaded on August 09,2023 at 11:58:59 UTC from IEEE Xplore. Restrictions apply.

The upper bound follows from Lemma 4. Therefore, $g = O((n/m^*)^{1/3})$. Substituting this bound on g into the upper bound on m and applying Lemma 1 gives the theorem

$$m = O((n/m^*)^{2/3}m^*) = O((n/\log n)^{2/3}m^*). \qquad \square$$

In the following sections, we describe three natural global algorithms. The preceding theorem provides an upper bound on the approximation ratio for all of them. In what follows, we establish a weak lower bound on the approximation ratio for each one individually.

E. LONGEST MATCH

Kieffer and Yang [5] proposed the LONGEST MATCH procedure, a global algorithm in which one always selects the *longest* maximal string. In our running example, the first rule added is $T \to abc\,abc$.

LONGEST MATCH has two elegant features that simplify analysis of its behavior.

1) No rule is ever introduced with a nonterminal on the right side.
2) Each nonterminal created appears in the final grammar.

If the first principle were violated, and a rule with a nonterminal X on the right was introduced, then the definition of the nonterminal X could not have been the longest maximal string when it was created, which contradicts the definition of the algorithm. The second principle follows from the first; since every new rule has only terminals on the right, nonterminals are only added to the grammar over the course the procedure and never eliminated.

The second principle offers a simple way to lower-bound the size of a grammar generated by LONGEST MATCH; we need only sum up the number of nonterminals created over the course of the procedure.

The first principle allows one to simplify the grammar maintained during the execution of LONGEST MATCH without altering the subsequent behavior of the algorithm. During the execution of LONGEST MATCH, we can replace each nonterminal on the right by a unique symbol. This does not alter subsequent behavior, since no rule containing a nonterminal will ever be introduced anyway. The example grammar from the start of this section can be transformed in this way into the following:

$$S \to |\,|\,ab\,a$$
$$T \to abc\,abc.$$

Furthermore, we can append the definitions of secondary rules to the end of the start rule (as long as they are separated by unique symbols), and then delete all secondary rules. Segments of the start rule that are between unique symbols can be rearranged within the string as well. Finally, we can delete unique symbols at the beginning and end of the start rule and merge consecutive unique symbols. Transforming the example in this way gives

$$S \to ab\,a \mid abc\,abc.$$

We refer to this three-step simplification procedure as *consolidating* a grammar. In analyzing the behavior of LONGEST MATCH on an input string, we are free to consolidate the grammar at any point to simplify analysis; the subsequent behavior of the procedure is unchanged.

1) Lower Bound:

Theorem 10: The approximation ratio of LONGEST MATCH is $\Omega(\log\log n)$.

Proof: We analyze the performance of LONGEST MATCH on a string σ_k, which consists of $k+2$ segments that are separated by uniques. First, define

$$\gamma_{[a,b]} = \begin{cases} x^{2^a}\,y^{2^{a+1}}\,x^{2^{a+2}}\,\cdots y^{2^b}, & \text{if } a \text{ is even} \\ y^{2^a}\,x^{2^{a+1}}\,y^{2^{a+2}}\,\cdots x^{2^b}, & \text{otherwise.} \end{cases}$$

For example, $\gamma_{[0,7]} = x\,y^2\,x^4\,y^8\,x^{16}\,y^{32}\,x^{64}\,y^{128}$.

Now, for i in the range 0 to $k-1$, define the ith segment of σ_k as $\gamma_{[i,i+2^j-1]}$ where j is the largest possible value such that $i+2^j \le k$. The final two segments are x^{2^k} and y^{2^k}. For example

$$\sigma_{10} = \gamma_{[0,7]} \mid \gamma_{[1,8]} \mid \gamma_{[2,9]} \mid \gamma_{[3,6]} \mid \gamma_{[4,7]} \mid \gamma_{[5,8]} \mid \gamma_{[6,9]} \mid \gamma_{[7,8]} \mid \gamma_{[8,9]} \mid \gamma_{[9,9]} \mid x^{1024} \mid y^{1024}$$

which expands to the following string (with indentation and line breaks added for clarity):

$$\begin{array}{llllllllll|}
x & y^2 & x^4 & y^8 & x^{16} & y^{32} & x^{64} & y^{128} & & \\
 & y^2 & x^4 & y^8 & x^{16} & y^{32} & x^{64} & y^{128} & x^{256} & \\
 & & x^4 & y^8 & x^{16} & y^{32} & x^{64} & y^{128} & x^{256} & y^{512} \\
 & & & y^8 & x^{16} & y^{32} & x^{64} & & & \\
 & & & & x^{16} & y^{32} & x^{64} & y^{128} & & \\
 & & & & & y^{32} & x^{64} & y^{128} & x^{256} & \\
 & & & & & & x^{64} & y^{128} & x^{256} & y^{512} \\
 & & & & & & & y^{128} & x^{256} & \\
 & & & & & & & & x^{256} & y^{512} \\
 & & & & & & & & & y^{512} \\
 & & & & & & & & & x^{1024} \\
 & & & & & & & & & y^{1024}
\end{array}$$

We analyze how LONGEST MATCH processes this string. Observe that in the example, the longest match is $\gamma_{[6,9]} = x^{64}\,y^{128}\,x^{256}\,y^{512}$. In general, the longest match in σ_k is always the *second* largest segment of the form $\gamma_{[a,k-1]}$. After this rule is added and the grammar rewritten, the next longest match is the third longest segment of the form $\gamma_{[a',k-1]}$ ($\gamma_{[8,9]}$ in our example), which is wholly contained in the first longest match. In the next round, the longest match is the fourth longest segment, and so forth. After $\log k$ rounds of this type, the next two longest matches are $x^{2^{k-1}}$ and $y^{2^{k-1}}$. At this point, the grammar is as follows (abbreviations introduced above are used for clarity):

$$\begin{aligned}
S &\to \gamma_{[0,7]} \mid \gamma_{[1,8]} \mid \gamma_{[2,5]}T_1 \mid \gamma_{[3,6]} \mid \gamma_{[4,7]} \mid \gamma_{[5,8]} \mid T_1 \mid \gamma_{[7,8]} \mid T_2 \mid T_3 \mid T_4T_4 \mid T_3T_3 \\
T_1 &\to x^{64}y^{128}T_2 \\
T_2 &\to x^{256}T_3 \\
T_3 &\to y^{512} \\
T_4 &\to x^{512}
\end{aligned}$$

and after consolidating the grammar, we obtain

$$S_2 \to \gamma_{[0,7]} \mid \gamma_{[1,8]} \mid \gamma_{[2,5]} \mid \gamma_{[3,6]} \mid \gamma_{[4,7]} \mid \gamma_{[5,8]} \mid \gamma_{[6,7]} \mid \gamma_{[7,8]} \mid \gamma_{[8,8]} \mid x^{512} \mid y^{512}.$$

Authorized licensed use limited to: Trial User - Warsaw University (Uniwersytet Warszawski). Downloaded on August 09,2023 at 11:58:59 UTC from IEEE Xplore. Restrictions apply.

The critical observation is that the consolidated grammar is the same as the initial grammar for input string σ_9. After another succession of rounds and a consolidation, the definition of the start rule becomes σ_8, and then σ_7, and so forth. Reducing the right side of the start rule from σ_i to σ_{i-1} entails the creation of at least $\lfloor \log i \rfloor$ nonterminals. Since nonterminals created by LONGEST MATCH are never eliminated, we can lower-bound the total size of the grammar produced on this input by

$$\sum_{i=1}^{k} \lfloor \log i \rfloor = \Omega(k \log k).$$

On the other hand, there exists a grammar of size $O(k)$ that generates σ_k. What follows is a sketch of the construction. First, we create nonterminals X_{2^i} and Y_{2^i} with expansions x^{2^i} and y^{2^i}, respectively, for all i up to k. We can define each such nonterminal using two symbols, and so only $O(k)$ symbols are required in total.

Then we define a nonterminal corresponding to each segment of σ_k. We define these nonterminals in batches, where a batch consists of all nonterminals corresponding to segments of σ_k that contain the same number of terms. Rather than describe the general procedure, we illustrate it with an example. Suppose that we want to define nonterminals corresponding to the following batch of segments in σ_{10}:

$$\gamma_{[3,6]} \mid \gamma_{[4,7]} \mid \gamma_{[5,8]} \mid \gamma_{[6,9]} \mid .$$

This can be done by defining the following auxiliary nonterminals which expand to prefixes and suffixes of the string $\gamma_{[3,9]} = y^8\, x^{16}\, y^{32}\, x^{64}\, y^{128}\, x^{256}\, y^{512}$:

$$\begin{aligned} P_1 &\to X_{64} & S_1 &\to Y_{128} \\ P_2 &\to Y_{32}P_1 & S_2 &\to S_1X_{256} \\ P_3 &\to X_{16}P_2 & S_3 &\to S_2Y_{512} \\ P_4 &\to Y_8P_3. \end{aligned}$$

Now we can define nonterminals corresponding to the desired segments $\gamma_{[i,j]}$ in terms of these "prefix" and "suffix" nonterminals as follows:

$$\begin{aligned} G_{[3,6]} &\to P_4 & G_{[4,7]} &\to P_3S_1 \\ G_{[5,8]} &\to P_2S_2 & G_{[6,9]} &\to P_1S_3. \end{aligned}$$

In this way, each nonterminal corresponding to a $\gamma_{[i,j]}$ in σ_k is defined using a constant number of symbols. Therefore, defining all k such nonterminals requires $O(k)$ symbols. We complete the grammar for σ_k by defining a start rule containing another $O(k)$ symbols. Thus, the total size of the grammar is $O(k)$.

Therefore, the approximation ratio for LONGEST MATCH is $\Omega(\log k)$. Since the length of σ_k is $n = \Theta(k2^k)$, this ratio is $\Omega(\log \log n)$ as claimed. □

F. GREEDY

Apostolico and Lonardi [11], [25], [10] proposed a variety of greedy algorithms for grammar-based data compression. The central idea, which we analyze here, is to select the maximal string that reduces the size of the grammar as much as possible. For example, on our usual starting grammar, the first rule added is $T \to abc$, since this decreases the size of the grammar by five symbols, which is the best possible.

Theorem 11: The approximation ratio of GREEDY is at least $(5 \log 3)/(3 \log 5) = 1.137\ldots$.

Proof: We consider the behavior of GREEDY on an input string of the form $\sigma_k = x^n$, where $n = 5^{2^k}$.

GREEDY begins with the grammar $S \to \sigma_k$. The first rule added must be of the form $T \to x^t$. The size of the grammar after this rule is added is then $t + \lfloor n/t \rfloor + (n \bmod t)$ where the first term reflects the cost of defining T, the second accounts for the instances of T itself, and the third represents extraneous x's. This sum is minimized when $t = n^{1/2}$. The resulting grammar is

$$\begin{aligned} S &\to T^{5^{2^{k-1}}} \\ T &\to x^{5^{2^{k-1}}}. \end{aligned}$$

Since the definitions of S and T contain no common symbols, we can analyze the behavior of GREEDY on each independently. Notice that both subproblems are of the same form as the original, but have size $k-1$ instead of k. Continuing in this way, we reach a grammar with 2^k nonterminals, each defined by five copies of another symbol. Each such rule is transformed as shown below in a final step that does not alter the size of the grammar

$$X \to YYYYY \Longrightarrow \begin{aligned} X &\to X'X'Y \\ X' &\to YY. \end{aligned}$$

Therefore, GREEDY generates a grammar for σ_k of size 5×2^k.

On the other hand, we show that for all n, x^n has a grammar of size $3\log_3(n) + o(\log n)$. Substituting $n = 5^{2^k}$ then proves the theorem.

Write n as a numeral in a base $b = 3^j$, where j is a parameter defined later: $n = d_0b^t + d_1b^{t-1} + d_2b^{t-2} + \cdots + d_{t-1}b^1 + d_t$.

The grammar is constructed as follows. First, create a nonterminal T_i with expansion x^i for each i between 1 and $b-1$. This can be done with $2 \cdot b = 2 \cdot 3^j$ symbols, using rules of the form $T_{i+1} \to T_ix$. Next, create a nonterminal U_0 with expansion x^{d_0} via the rule $U_0 \to T_{d_0}$. Create a nonterminal U_1 with expansion $x^{d_0b+d_1}$ by introducing intermediate nonterminals Z_1 which triples U_0, Z_2 which triples Z_1, and so on j times, and then by appending T_{d_1}

$$\begin{aligned} Z_1 &\to U_0U_0U_0 \\ Z_2 &\to Z_1Z_1Z_1 \\ &\cdots \\ Z_j &\to Z_{j-1}Z_{j-1}Z_{j-1} \\ U_1 &\to Z_jT_{d_1}. \end{aligned}$$

This requires $3j+2$ symbols. Similarly, create U_2 with expansion $x^{d_0b^2+d_1b+d_2}$, and so on. The start symbol of the grammar is U_t. The total number of symbols used is at most

$$\begin{aligned} 2 \cdot 3^j + (3j+2) \cdot t &= 2 \cdot 3^j + (3j+2) \cdot \log_b n \\ &= 2 \cdot 3^j + 3\log_3 n + \frac{2}{j}\log_3 n. \end{aligned}$$

The second equality uses the fact that $b = 3^j$. Setting $j = (1/2)\log_3 \log_3 n$ makes the last expression $3\log_3(n) + o(\log n)$ as claimed. □

Authorized licensed use limited to: Trial User - Warsaw University (Uniwersytet Warszawski). Downloaded on August 09,2023 at 11:58:59 UTC from IEEE Xplore. Restrictions apply.

G. RE-PAIR

Larsson and Moffat [7] proposed the RE-PAIR algorithm. (The byte-pair encoding (BPE) technique of Gage [34] is based on similar ideas.) Essentially, this is a global algorithm in which the maximal string chosen at each step is the one which appears most often. In our running example, the first rule added is $T \to ab$, since the string ab appears most often.

There is a small difference between the algorithm originally proposed by Larsson and Moffat and what we refer to here as RE-PAIR: the original algorithm always makes a rule for the pair of symbols that appears most often without overlap, regardless of whether that pair forms a maximal string. For example, on input $xyzxyz$, the original RE-PAIR algorithm generates the following grammar:

$$S \to UU \quad U \to xV \quad V \to yz$$

This is unattractive, since one could replace the single occurrence of the nonterminal V by its definition and obtain a smaller grammar. Indeed, RE-PAIR, as described here, would give the smaller grammar:

$$S \to UU \quad U \to xyz.$$

The original approach was motivated by implementation efficiency issues.

Theorem 12: The approximation ratio of RE-PAIR is $\Omega(\sqrt{\log n})$.

Proof: Consider the performance of RE-PAIR on input strings of the form

$$\sigma_k = \prod_{w=\sqrt{k}}^{2\sqrt{k}} \prod_{i=0}^{w-1} (x^{b_{w,i}}\ |)$$

where $b_{w,i}$ is an integer that, when written as a k-bit binary number, has a 1 at each position j such that $j \equiv i \pmod w$. (Position 0 corresponds to the least significant bit.) On such an input, RE-PAIR creates rules for strings of x's with lengths that are powers of two: $X_1 \to xx, X_2 \to X_1X_1 \ldots$.

At this point, each run of x's with length $b_{w,i}$ in σ_k is represented using one nonterminal for each 1 in the binary representation of $b_{w,i}$. For example, the beginning of σ_{16} and the beginning of the resulting start rule are listed as follows:

$$\begin{array}{rlll}
\sigma_{16} = x^{0001000100010001} & | & x^{0010001000100010} & | \\
x^{0100010001000100} & | & x^{1000100010001000} & | \\
x^{1000100010001001} & | & x^{0001000010000010} & | \quad \ldots \\
S \to X_{12}X_8X_4x & | & X_{13}X_9X_5X_1 & | \\
X_{14}X_{10}X_6X_2 & | & X_{15}X_{11}X_7X_3 & | \\
X_{15}X_{10}X_5x & | & X_{11}X_6X_1 & | \quad \ldots
\end{array}$$

Note that no other rules are introduced, because each pair of adjacent symbols now appears only once. RE-PAIR encodes each string of x's using $\Omega(\sqrt{k})$ symbols. Since there are $\Omega(k)$ such strings, the size of the grammar produced is $\Omega(k^{3/2})$.

On the other hand, there exists a grammar of size $O(k)$ that generates σ_k. First, we create a nonterminal X_j with expansion x^{2^j} for all j up to $k-1$. This requires $O(k)$ symbols. Then for each w, we create a nonterminal $B_{w,0}$ for $x^{b_{w,0}}$ using $O(\sqrt{k})$ of the X_j nonterminals, just as RE-PAIR does. However, we can then define a nonterminal for each remaining string of x's using only two symbols: $B_{w,i} \to B_{w,i-1}B_{w,i-1}$, for a total of $O(k)$ additional symbols. Finally, we expend $O(k)$ symbols on a start rule, which consists of all the $B_{w,i}$ separated by unique symbols. In total, the grammar size is $O(k)$ as claimed.

To complete the argument, note that $n = |\sigma_k| = \Theta(\sqrt{k}2^k)$, and so the approximation ratio is no better than $\Omega(\sqrt{k}) = \Omega(\sqrt{\log n})$. □

VII. NEW ALGORITHMS

In this section, we present a simple $O(\log^3 n)$ approximation algorithm for the smallest grammar problem. We then give a more complex algorithm with approximation ratio $O(\log n/m^*)$ based on an entirely different approach.

A. An $O(\log^3 N)$ Approximation Algorithm

To begin, we describe a useful grammar construction, prove one lemma, and cite an old result that we shall use later.

The *substring construction* generates a set of grammar rules enabling each substring of a string $\eta = x_1 \ldots x_p$ to be expressed with at most two symbols.

The construction works as follows. First, create a nonterminal for each suffix of the string $x_1 \ldots x_k$ and each prefix of $x_{k+1} \ldots x_p$, where $k = \lceil (p/2) \rceil$. Note that each such nonterminal can be defined using only two symbols: the nonterminal for the next shorter suffix or prefix together with one symbol x_i. Repeat this construction recursively on the two halves of the original string $x_1 \ldots x_k$ and $x_{k+1} \ldots x_p$. The recursion terminates when a string of length one is obtained. This recursion has $\log p$ levels, and p nonterminals are defined at each level. Since each definition contains at most two symbols, the total cost of the construction is at most $2p \log p$.

Now we show that every substring $\alpha = x_i \ldots x_j$ of η is equal to $\langle AB \rangle$, where A and B are nonterminals defined in the construction. There are two cases to consider. If α appears entirely within the left-half of η or entirely within the right-half, then we can obtain A and B from the recursive construction on $x_1 \ldots x_k$ or $x_{k+1} \ldots x_p$. Otherwise, let $k = \lceil (p/2) \rceil$ as before, and let A be the nonterminal for $x_i \ldots x_k$, and let B be the nonterminal for $x_{k+1} \ldots x_j$.

For example, the substring construction for the string $\eta = abcdefgh$ is given as follows:

$$\begin{array}{llll}
C_1 \to d & C_2 \to cC_1 & C_3 \to bC_2 & C_4 \to aC_3 \\
D_1 \to e & D_2 \to D_1f & D_3 \to D_2g & D_4 \to D_3h \\
E_1 \to b & E_2 \to aE_1 & F_1 \to c & F_2 \to F_1d \\
G_1 \to f & G_2 \to eG_1 & H_1 \to g & H_2 \to H_1h.
\end{array}$$

With these rules defined, each substring of $abcdefgh$ is expressible with at most two symbols. For example, $defg = \langle C_1D_3 \rangle$. In the next lemma, we present a variation of Lemma 3 needed for the new algorithm.

Authorized licensed use limited to: Trial User - Warsaw University (Uniwersytet Warszawski). Downloaded on August 09,2023 at 11:58:59 UTC from IEEE Xplore. Restrictions apply.

Lemma 8: Let σ be a string generated by a grammar of size m. Then there exists a string β_k of length at most $2mk$ that contains every length-k substring of σ.

Proof: We can construct β_k by concatenating strings obtained from the rules of the grammar of size m. For each rule, $T \rightarrow \alpha$, do the following.

1) For each terminal in α, take the length-k substring of $\langle T \rangle$ beginning at that terminal.
2) For each nonterminal in α, take the length-$(2k-1)$ substring of $\langle T \rangle$ consisting of the last character in the expansion of that nonterminal, the preceding $k-1$ characters, and the following $k-1$ characters.

In both cases, we permit the substrings to be shorter if they are truncated by the start or end of $\langle T \rangle$.

Now we establish the correctness of this construction. First, note that the string β_k is a concatenation of at most m strings of length at most $2k$, giving a total length of at most $2mk$ as claimed. Next, let γ be a length-k substring of σ. Consider the rule $T \rightarrow \alpha$ such that $\langle T \rangle$ contains γ and $\langle T \rangle$ is as short as possible. Either γ begins at a terminal of α, in which case it is a string of type 1), or else it begins inside the expansion of a nonterminal in α and ends beyond, in which case it is contained in a string of type 2). (Note that γ cannot be wholly contained in the expansion of a nonterminal in α; otherwise, we would have selected that nonterminal for consideration instead of T.) In either case, γ is a substring of β_k as desired. □

Our approximation algorithm for the smallest grammar problem makes use of Blum, Jiang, Li, Tromp, and Yannakakis' 4-approximation for the shortest superstring problem [35]. In this procedure, we are given a collection of strings and want to find the shortest superstring; that is, the shortest string that contains each string in the collection as a substring. The procedure works greedily. At each step, find the two strings in the collection with largest overlap. Merge these two into a single string. (For example, $abaa$ and $aaac$ have overlap aa and thus can be merged to form $abaaac$.) Repeat this process until only one string remains. This is the desired superstring, and Blum *et al.* proved that it is at most four times longer than the shortest superstring.

B. The Algorithm

In this algorithm, the focus is on certain sequences of substrings of σ. In particular, we construct $\log n$ sequences $C_n, C_{n/2}, C_{n/4}, \ldots, C_2$, where the sequence C_k consists of some substrings of σ that have length at most k. These sequences are defined as follows. The sequence C_n is initialized to consist of only the string σ itself. In general, the sequence C_k generates the sequence $C_{k/2}$ via the following operations, which are illustrated in Fig. 1.

1) Use the greedy 4-approximation algorithm of Blum *et al.* to form a superstring ρ_k containing all the distinct strings in C_k.
2) Cut the superstring ρ_k into small pieces. First, determine where each string in C_k ended up inside ρ_k, and then cut ρ_k at the left endpoints of those strings.
3) Cut each piece of ρ_k that has length greater than $k/2$ at the midpoint. During the analysis, we shall refer to the cuts made during this step as *extra cuts*.

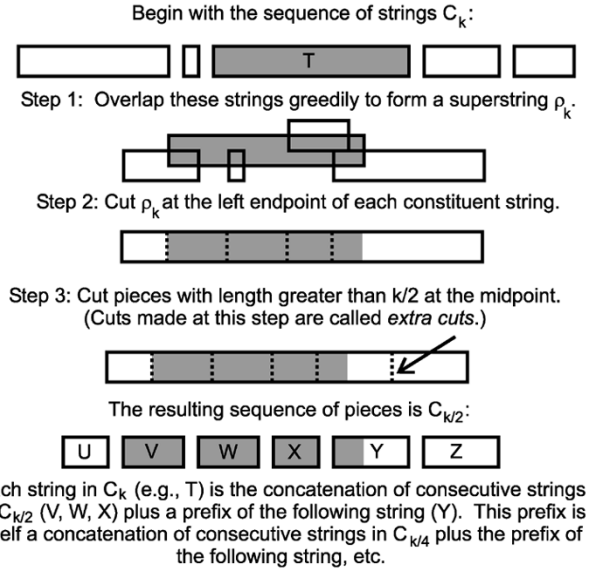


Fig. 1.

The sequence $C_{k/2}$ is defined to be the sequence of pieces of ρ_k generated by this three-step process. By the nature of Blum's algorithm, no piece of ρ_k can have length greater than k after step 2), and so no piece can have length greater than $k/2$ after step 3). Thus, $C_{k/2}$ is a sequence of substrings of σ that have length at most $k/2$ as desired.

Now we translate these sequences of strings into a grammar. To begin, associate a nonterminal with each string in each sequence C_k. In particular, the nonterminal associated with the single string in C_n (which is σ itself) is the start symbol of the grammar.

All that remains is to define these nonterminals. In doing so, the following observation is key: each string in C_k is the concatenation of several consecutive strings in $C_{k/2}$ together with a prefix of the next string in $C_{k/2}$. This is illustrated in the figure above, where the fate of one string in C_k (shaded and marked T) is traced through the construction of $C_{k/2}$. In this case, T is the concatenation of V, W, X, and a prefix of Y. Similarly, the prefix of Y is itself the concatenation of consecutive strings in $C_{k/4}$ together with a prefix of the next string in $C_{k/4}$. This prefix is, in turn, the concatenation of consecutive strings in $C_{k/8}$ together with a prefix of the next string in $C_{k/8}$, etc. As a result, we can define the nonterminal corresponding to a string in C_k as a sequence of consecutive nonterminals from $C_{k/2}$, followed by consecutive nonterminals from $C_{k/4}$, followed by consecutive nonterminals from $C_{k/8}$, etc. For example, the definition of T would begin $T \rightarrow VWX\ldots$ and then contain sequences of consecutive nonterminals from $C_{k/4}, C_{k/8}$, etc. As a special case, the nonterminals corresponding to strings in C_2 can be defined in terms of terminals.

We can use the substring construction to make these definitions shorter and hence the overall size of the grammar smaller.

Authorized licensed use limited to: Trial User - Warsaw University (Uniwersytet Warszawski). Downloaded on August 09,2023 at 11:58:59 UTC from IEEE Xplore. Restrictions apply.

In particular, for each sequence of strings C_k, we apply the substring construction on the corresponding sequence of nonterminals. This enables us to express any sequence of consecutive nonterminals using just two symbols. As a result, we can define each nonterminal corresponding to a string in C_k using only two symbols that represent a sequence of consecutive nonterminals from $C_{k/2}$, two more that represent a sequence of consecutive nonterminals from $C_{k/4}$, etc. Thus, every nonterminal can now be defined with $O(\log n)$ symbols on the right.

Theorem 13: The procedure described above is an $O(\log^3 n)$-approximation algorithm for the smallest grammar problem.

Proof: We must determine the size of the grammar generated by the above procedure. In order to do this, we must first upper-bound the number of strings in each sequence C_k. To this end, note that the number of strings in $C_{k/2}$ is equal to the number of strings in C_k plus the number of extra cuts made in step 3). Thus, given that C_n contains a single string, we can upper-bound the number of strings in C_k by upper-bounding the number of extra cuts made at each stage.

Suppose that the smallest grammar generating σ has size m^*. Then Lemma 8 implies that there exists a superstring containing all the strings in C_k with length $2m^*k$. Since we are using a 4-approximation, the length of ρ_k is at most $8m^*k$. Therefore, there can be at most $16m^*$ pieces of ρ_k with length greater than $k/2$ after step 2). This upper-bounds the number of extra cuts made in the formation of $C_{k/2}$, since extra cuts are only made into pieces with length greater than $k/2$. It follows that every sequence of strings C_k has length $O(m^* \log n)$, since step 2) is repeated only $\log n$ times over the course of the algorithm.

On one hand, there are $\log n$ sequences C_k, each containing $O(m^* \log n)$ strings. Each such string corresponds to a nonterminal with a definition of length $O(\log n)$. This gives $O(m^* \log^3 n)$ symbols in total. On the other hand, for each sequence of strings C_k, we apply the substring construction on the corresponding sequence of nonterminals. Recall that this construction generates $2p \log p$ symbols when applied to a sequence of length p. This creates an additional

$$O((\log n) \cdot (m^* \log n) \log(m^* \log n)) = O(m^* \log^3 n)$$

symbols. Therefore, the total size of the grammar generated by this algorithm is $O(m^* \log^3 n)$, which proves the claim. □

C. An $O(\log n/m^*)$-Approximation Algorithm

We now present a more complex solution to the smallest grammar problem with approximation ratio $O(\log n/m^*)$. The description is divided into three sections. First, we introduce a variant of the well-known LZ77 compression scheme. This serves two purposes: it gives a new lower bound on the size of the smallest grammar for a string and is the starting point for our construction of a small grammar. Second, we introduce balanced binary grammars, the variety of well-behaved grammars that our procedure employs. In the same section, we also introduce three basic operations on balanced binary grammars. Finally, we present the main algorithm, which translates a string compressed using our LZ77 variant into a grammar at most $O(\log n/m^*)$ times larger than the smallest.

D. An LZ77 Variant

We begin by describing a variant of LZ77 compression [21]. We use this both to obtain a lower bound on the size of the smallest grammar for a string and as the basis for generating a small grammar. In this scheme, a string is represented by a sequence of characters and pairs of integers. For example, one possible sequence is

$$a\ b\ (1,2)\ (2,3)\ c\ (1,5)$$

An LZ77 representation can be decoded into a string by working left-to-right through the sequence according to the following rules.

- If a character c is encountered in the sequence, then the next character in the string is c.
- Otherwise, if a pair (x, y) is encountered in the sequence, then the next y characters of the string are the same as the y characters beginning at position x of the string. (We require that the y characters beginning at position x be represented by earlier items in the sequence.)

The example sequence can be decoded as follows:

Index:	1	2	3	4	5	6	7	8	9	10	11	12	13
LZ77:	a	b	$(1,2)$		$(2,3)$			c	$(1,5)$				
String:	a	b	a	b	b	a	b	c	a	b	a	b	b

The shortest LZ77 sequence for a given string can be found in polynomial time. Make a left-to-right pass through the string. If the next character in the unprocessed portion of the string has not appeared before, output it. Otherwise, find the longest prefix of the unprocessed portion that appears in the processed portion and output the pair (x, y) describing that previous appearance. It is easy to show (and well known) that this procedure finds the shortest LZ77 sequence.

The following lemma states that this procedure implies a lower bound on the size of the smallest grammar.

Lemma 9: The length of the shortest LZ77 sequence for a string is a lower bound on the size of the smallest grammar for that string.

Proof: Suppose that a string is generated by a grammar of size m^*. We can transform this grammar into an LZ77 sequence of length at most m^* as follows. Begin with the sequence of symbols on the right side of the start rule. Select the nonterminal with longest expansion. Replace the leftmost instance by its definition and replace each subsequent instance by a pair referring to the first instance. Repeat this process until no nonterminals remain. Note that each symbol on the right side of the original grammar corresponds to at most one item in the resulting sequence. This establishes the desired inequality. □

A somewhat similar process was described in [36]. In contrast, our $O(\log n/m^*)$-approximation algorithm essentially inverts the process and maps an LZ77 sequence to a grammar. This other direction is much more involved.

E. Balanced Binary Grammars

In this subsection, we introduce the notion of a balanced binary grammar. The approximation algorithm we are developing

Authorized licensed use limited to: Trial User - Warsaw University (Uniwersytet Warszawski). Downloaded on August 09,2023 at 11:58:59 UTC from IEEE Xplore. Restrictions apply.

works exclusively with this restricted class of well-behaved grammars.

A *binary rule* is a grammar rule with exactly two symbols on the right side. A *binary grammar* is a grammar in which every rule is binary. Two strings of symbols, β and γ, are α-*balanced* if

$$\frac{\alpha}{1-\alpha} \leq \frac{[\beta]}{[\gamma]} \leq \frac{1-\alpha}{\alpha}$$

for some constant α between 0 and $\frac{1}{2}$. Intuitively, α-balanced means "about the same length." Note that inverting the fraction $\frac{[\beta]}{[\gamma]}$ gives an equivalent condition. An α-*balanced rule* is a binary rule in which the two symbols on the right are α-balanced. An α-*balanced grammar* is a binary grammar in which every rule is α-balanced. For brevity, we use "balanced" to signify "α-balanced."

The remainder of this subsection defines three basic operations on balanced binary grammars: ADDPAIR, ADDSEQUENCE, and ADDSUBSTRING. Each operation adds a small number of rules to an existing balanced grammar to produce a new balanced grammar that has a nonterminal with specified properties. For these operations to work correctly, we require that α be selected from the limited range $0 < \alpha \leq 1 - \frac{1}{2}\sqrt{2}$, which is about 0.293. These three operations are detailed in the following paragraphs.

1) The ADDPAIR Operation: This operation begins with a balanced grammar containing symbols X and Y and produces a balanced grammar containing a nonterminal with expansion $\langle XY \rangle$. The number rules added to the original grammar is

$$O\left(1 + \left|\log \frac{[X]}{[Y]}\right|\right).$$

Suppose that $[X] \leq [Y]$; the other case is symmetric. The ADDPAIR operation is divided into two phases.

In the first phase, we decompose Y into a string of symbols. Initially, this string consists of the symbol Y itself. Thereafter, while the first symbol in the string is not in balance with X, we replace it by its definition. A routine calculation, which we omit, shows that balance is eventually achieved. At this point, we have a string of symbols $Y_1, \ldots, Y_t$ with expansion $\langle Y \rangle$ such that Y_1 is in balance with X. Furthermore, note that $Y_1, \ldots, Y_i$ is in balance with Y_{i+1} for all $1 \leq i < t$ by construction.

In the second phase, we build a balanced binary grammar for the following sequence of nonterminals generated during the first phase:

$$X\ Y_1\ Y_2\ \ldots\ Y_t.$$

The analysis of the second phase runs for many pages, even though we omit some routine algebra. Initially, we create a new rule $Z_1 \rightarrow XY_1$ and declare this to be the *active rule*. The remainder of the second phase is divided into steps. At the start of the ith step, the active rule has the form $Z_i \rightarrow A_iB_i$, and the following three invariants hold.

(B1) $\langle Z_i \rangle = \langle XY_1 \ldots Y_i \rangle$.
(B2) $\langle B_i \rangle$ is a substring of $\langle Y_1, \ldots, Y_i \rangle$.
(B3) All rules in the grammar are balanced, including the active rule.

The relationships between strings implied by the first two invariants are indicated in the following diagram:

$$\overbrace{\underbrace{X\ Y_1\ Y_2\ \ldots}_{A_i}\ \ldots\ \underbrace{Y_{i-1}\ Y_i}_{B_i}}^{Z_i} Y_{i+1}\ \ldots\ Y_t.$$

After t steps, the active rule defines a nonterminal Z_t with expansion $\langle XY_1 \ldots Y_t \rangle = \langle XY \rangle$ as desired, completing the procedure.

The invariants stated above imply some inequalities that are needed later to show that the grammar remains in balance. Since $Y_1 \ldots Y_i$ is in balance with Y_{i+1}, we have

$$\frac{\alpha}{1-\alpha} \leq \frac{[Y_{i+1}]}{[Y_1 \ldots Y_i]} \leq \frac{1-\alpha}{\alpha}.$$

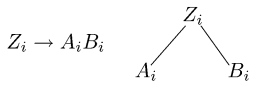
Since $\langle B_i \rangle$ is a substring of $\langle Y_1 \ldots Y_i \rangle$ by invariant (B2), we can conclude

$$\frac{\alpha}{1-\alpha} \leq \frac{[Y_{i+1}]}{[B_i]}. \tag{1}$$

On the other hand, since $\langle Z_i \rangle$ is a superstring of $\langle Y_1 \ldots Y_i \rangle$ by invariant (B1), we can conclude

$$\frac{[Y_{i+1}]}{[Z_i]} \leq \frac{1-\alpha}{\alpha}. \tag{2}$$

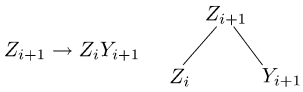
All that remains is to describe how each step of the second phase is carried out. Each step involves intricate grammar transformations, and so for clarity, we supplement the text with diagrams. In these diagrams, a rule $Z_i \rightarrow A_iB_i$ is indicated with a wedge.



Pre-existing rules are indicated with shaded lines, and new rules with dark lines.

At the start of the ith step, the active rule is $Z_i \rightarrow A_iB_i$. Our goal is to create a new active rule that defines Z_{i+1} while maintaining the three invariants. There are three cases to consider.

Case 1: If Z_i and Y_{i+1} are in balance, then we create a new rule.



This becomes the active rule. It is easy to check that the three invariants are maintained.

If Z_i and Y_{i+1} are not in balance, this implies that

$$\frac{\alpha}{1-\alpha} \leq \frac{[Y_{i+1}]}{[Z_i]} \leq \frac{1-\alpha}{\alpha}$$

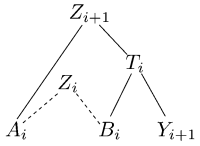
does not hold. Since the right inequality is (2), the left inequality must be violated. Thus, hereafter we can assume

$$\frac{\alpha}{1-\alpha} > \frac{[Y_{i+1}]}{[Z_i]}. \tag{3}$$

Case 2: Otherwise, if A_i is in balance with B_iY_{i+1}, then we create two new rules.

Authorized licensed use limited to: Trial User - Warsaw University (Uniwersytet Warszawski). Downloaded on August 09,2023 at 11:58:59 UTC from IEEE Xplore. Restrictions apply.

$$\begin{aligned} Z_{i+1} &\rightarrow A_i T_i \\ T_i &\rightarrow B_i Y_{i+1} \end{aligned}$$



The first of these becomes the active rule. It is easy to check that the first two invariants are maintained. In order to check that all new rules are balanced, first note that the rule $Z_{i+1} \rightarrow A_i T_i$ is balanced by the case assumption. For the rule $T_i \rightarrow B_i Y_{i+1}$ to be balanced, we must show

$$\frac{\alpha}{1-\alpha} \leq \frac{[Y_{i+1}]}{[B_i]} \leq \frac{1-\alpha}{\alpha}.$$

The left inequality is (1). For the right inequality, begin with (3)

$$\begin{aligned} [Y_{i+1}] &< \frac{\alpha}{1-\alpha}[Z_i] = \frac{\alpha}{1-\alpha}([A_i]+[B_i]) \\ &\leq \frac{\alpha}{1-\alpha}\left(\frac{1-\alpha}{\alpha}[B_i]+[B_i]\right) \\ &\leq \frac{1-\alpha}{\alpha}[B_i]. \end{aligned}$$

The equality follows from the definition of Z_i by the rule $Z_i \rightarrow A_i B_i$. The subsequent inequality uses the fact that this rule is balanced, according to invariant (B3). The last inequality uses only algebra and holds for all $\alpha \leq 0.381$.

If case 2 is bypassed, then A_i and $B_i Y_{i+1}$ are not in balance which implies that

$$\frac{\alpha}{1-\alpha} \leq \frac{[A_i]}{[B_i Y_{i+1}]} \leq \frac{1-\alpha}{\alpha}$$

does not hold. Since A_i is in balance with B_i alone by invariant (B3), the right inequality holds. Therefore, the left inequality must not; hereafter, we can assume

$$\frac{\alpha}{1-\alpha} > \frac{[A_i]}{[B_i Y_{i+1}]}. \tag{4}$$

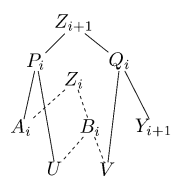
Combining inequalities (3) and (4), one can use algebraic manipulation to establish the following bounds, which hold hereafter:

$$\frac{[A_i]}{[B_i]} \leq \frac{\alpha}{1-2\alpha} \tag{5}$$

$$\frac{[Y_{i+1}]}{[B_i]} \leq \frac{\alpha}{1-2\alpha}. \tag{6}$$

Case 3: Otherwise, suppose that B_i is defined by the rule $B_i \rightarrow UV$. We create three new rules.

$$\begin{aligned} Z_{i+1} &\rightarrow P_i Q_i \\ P_i &\rightarrow A_i U \\ Q_1 &\rightarrow V Y_{i+1} \end{aligned}$$



The first of these becomes the active rule. We must check that all of the new rules are in balance. We begin with $P_i \rightarrow A_i U$. In one direction, we have

$$\frac{[A_i]}{[U]} \geq \frac{[A_i]}{(1-\alpha)[B_i]} \geq \frac{[A_i]}{[B_i]} \geq \frac{\alpha}{1-\alpha}.$$

The first inequality uses the fact that $B_i \rightarrow UV$ is balanced. The second inequality follows because $1-\alpha \leq 1$. The final inequality uses the fact that A_i and B_i are in balance. In the other direction, we have

$$\frac{[A_i]}{[U]} \leq \frac{[A_i]}{\alpha[B_i]} \leq \frac{1}{1-2\alpha} \leq \frac{1-\alpha}{\alpha}.$$

The first inequality uses the fact that $B_i \rightarrow UV$ is balanced, and the second follows from (5). The last inequality holds for all $\alpha < 0.293$. The argument to show that $Q_i \rightarrow V Y_{i+1}$ is balanced is similar.

Finally, we must check that $Z_{i+1} \rightarrow P_i Q_i$ is in balance. In one direction, we have

$$\begin{aligned} \frac{[P_i]}{[Q_i]} = \frac{[A_i U]}{[V Y_{i+1}]} &\leq \frac{[A_i]+(1-\alpha)[B_i]}{\alpha[B_i]+[Y_{i+1}]} = \frac{\frac{[A_i]}{[B_i]}+(1-\alpha)}{\alpha+\frac{[Y_{i+1}]}{[B_i]}} \\ &\leq \frac{\frac{\alpha}{1-2\alpha}+(1-\alpha)}{\alpha+\frac{\alpha}{1-\alpha}} \\ &\leq \frac{1-\alpha}{\alpha}. \end{aligned}$$

The equality follows from the definitions of P_i and Q_i. The first inequality uses the fact that the rule $B_i \rightarrow UV$ is balanced. The subsequent equality follows by dividing the top and bottom by $[B_i]$. In the next step, we use (5) on the top, and (1) on the bottom. The final inequality holds for all $\alpha \leq (1/3)$. In the other direction, we have

$$\begin{aligned} \frac{[P_i]}{[Q_i]} = \frac{[A_i U]}{[V Y_{i+1}]} &\geq \frac{[A_i]+\alpha[B_i]}{(1-\alpha)[B_i]+[Y_{i+1}]} \\ &= \frac{\frac{[A_i]}{[B_i]}+\alpha}{(1-\alpha)+\frac{[Y_{i+1}]}{[B_i]}} \\ &\geq \frac{\frac{\alpha}{1-\alpha}+\alpha}{(1-\alpha)+\frac{\alpha}{1-2\alpha}} \\ &\geq \frac{\alpha}{1-\alpha}. \end{aligned}$$

As before, the first inequality uses the definitions of P_i and Q_i. Then we use the fact that $B_i \rightarrow UV$ is balanced. We obtain the second equality by dividing the top and bottom by $[B_i]$. The subsequent inequality uses the fact that A_i and B_i are in balance on the top and (6) on the bottom. The final inequality holds for all $\alpha \leq (1/3)$.

All that remains is to upper-bound the number of rules created during the ADDPAIR operation. At most three rules are added in each of the t steps of the second phase. Therefore, it suffices to upper-bound t. This quantity is determined during the first phase, where Y is decomposed into a string of symbols. In each step of the first phase, the length of the expansion of the first symbol in this string decreases by a factor of at least $1-\alpha$. When the first symbol is in balance with X, the process stops. Therefore, the number of steps is $O(\log[Y]/[X])$. Since the string initially contains one symbol, t is $O(1+\log[Y]/[X])$. Therefore, the number of new rules is

$$O\left(1+\left|\log\frac{[X]}{[Y]}\right|\right).$$

Because we take the absolute value, this bound holds regardless of whether $[X]$ or $[Y]$ is larger.

Authorized licensed use limited to: Trial User - Warsaw University (Uniwersytet Warszawski). Downloaded on August 09,2023 at 11:58:59 UTC from IEEE Xplore. Restrictions apply.

2) The ADDSEQUENCE Operation: The ADDSEQUENCE operation is a generalization of ADDPAIR. Given a balanced grammar with symbols $X_1 \dots X_t$, the operation creates a balanced grammar containing a nonterminal with expansion $\langle X_1 \dots X_t \rangle$. The number of rules added is

$$O\left(t\left(1+\log\frac{[X_1\dots X_t]}{t}\right)\right).$$

The idea is to place the X_i at the leaves of a balanced binary tree. (To simplify the analysis, assume that t is a power of two.) We create a nonterminal for each internal node by combining the nonterminals at the child nodes using ADDPAIR. Recall that the number of rules that ADDPAIR creates when combining nonterminals X and Y is

$$O\left(1+\left|\log\frac{[X]}{[Y]}\right|\right) = O(\log[X]+\log[Y]).$$

Let c denote the hidden constant on the right, and let s equal $[X_1 \dots X_t]$. Creating all the nonterminals on the bottom level of the tree generates at most

$$c\sum_{i=1}^{t}\log[X_i] \le ct\log\frac{s}{t}$$

rules. (The inequality follows from the concavity of log.) Similarly, the number of rules created on the second level of the tree is at most $c(t/2)\log\frac{s}{t/2}$, because we pair $t/2$ nonterminals, but the sum of their expansion lengths is still s. In general, on the ith level, we create at most

$$c(t/2^i)\log\frac{s}{t/2^i} = c(t/2^i)\log\frac{s}{t} + cti/2^i$$

new rules. Summing i from 0 to $\log t$, we find that the total number of rules created is

$$\sum_{i=0}^{\log t} c(t/2^i)\log\frac{s}{t} + cti/2^i = O\left(t\left(1+\log\frac{[X_1\dots X_t]}{t}\right)\right)$$

as claimed.

3) The ADDSUBSTRING Operation: This operation takes a balanced grammar containing a nonterminal with β as a substring and produces a balanced grammar containing a nonterminal with expansion exactly β while adding $O(\log|\beta|)$ new rules.

Let T be the nonterminal with the shortest expansion such that its expansion contains β as a substring. Let $T \to XY$ be its definition. Then we can write $\beta = \beta_p\beta_s$, where the prefix β_p lies in $\langle X\rangle$ and the suffix β_s lies in $\langle Y\rangle$. (Note, β_p is actually a suffix of $\langle X\rangle$, and β_s is a prefix of $\langle Y\rangle$.) We generate a nonterminal that expands to the prefix β_p, another that expands to the suffix β_s, and then merge the two with ADDPAIR. The last step generates only $O(\log|\beta|)$ new rules. So all that remains is to generate a nonterminal that expands to the prefix β_p; the suffix is handled symmetrically. This task is divided into two phases.

In the first phase, we find a sequence of symbols $X_1 \dots X_t$ with expansion equal to β_p. To do this, we begin with an empty sequence and employ a recursive procedure. At each step, we have a *desired suffix* (initially β_p) of some *current symbol* (initially X). During each step, we consider the definition of the current symbol, say $X \to AB$. There are two cases.

1) If the desired suffix wholly contains $\langle B\rangle$, then we prepend B to the nonterminal sequence. The desired suffix becomes the portion of the old suffix that overlaps $\langle A\rangle$, and the current nonterminal becomes A.
2) Otherwise, we keep the same desired suffix, but the current symbol becomes B.

A nonterminal is only added to the sequence in case 1). But in that case, the length of the desired suffix is scaled down by at least a factor $1-\alpha$. Therefore, the length of the resulting nonterminal sequence is $t = O(\log|\beta|)$.

This construction implies the following inequality, which we use later:

$$\frac{[X_1\dots X_i]}{[X_{i+1}]} \le \frac{1-\alpha}{\alpha}. \tag{7}$$

This inequality holds because $\langle X_1 \dots X_i\rangle$ is a suffix of the expansion of a nonterminal in balance with X_{i+1}. Consequently, $X_1 \dots X_i$ is not too long to be in balance with X_{i+1}.

In the second phase, we merge the nonterminals in the sequence $X_1 \dots X_t$ to obtain the nonterminal with expansion β_p. The process goes from left to right. Initially, we set $R_1 = X_1$. Thereafter, at the start of the ith step, we have a nonterminal R_i with expansion $\langle X_1 \dots X_i\rangle$ and seek to merge in symbol X_{i+1}. There are two cases, distinguished by whether or not the following inequality holds:

$$\frac{\alpha}{1-\alpha} \le \frac{[R_i]}{[X_{i+1}]}.$$

- If so, then R_i and X_{i+1} are in balance. (Inequality (7) supplies the needed upper bound on $[R_i]/[X_{i+1}]$.) Therefore, we add the rule $R_{i+1} \to R_iX_{i+1}$.
- If not, then R_i is too small to be in balance with X_{i+1}. (It cannot be too large, because of inequality (7).) We use ADDPAIR to merge the two, which generates $O(1+\log[X_{i+1}]/[R_i])$ new rules. Since $[R_i]$ is at most a constant times the size of its largest component $[X_i]$, the number of new rules is $O(1+\log[X_{i+1}]/[X_i])$.

Summing the number of rules created during this process gives

$$\sum_{i=1}^{t} O\left(1+\log\frac{[X_{i+1}]}{[X_i]}\right) = O(t+\log[X_t]) = O(\log|\beta|).$$

The second equality follows from the fact, observed previously, that $t = O(\log|\beta|)$ and from the fact that $\langle X_t\rangle$ is a substring of β. Generating a nonterminal for the suffix β_s requires $O(\log|\beta|)$ rules as well. Therefore, the total number of new rules is $O(\log|\beta|)$ as claimed.

F. The Algorithm

We now combine all the tools of the preceding two sections to obtain an $O(\log n/m^*)$-approximation algorithm for the smallest grammar problem.

We are given an input string σ. First, we apply the LZ77 variant described in Section VII-D. This gives a sequence $L_1 \dots L_p$ of terminals and pairs. By Lemma 9, the length of this sequence is a lower bound on the size of the smallest grammar for σ; that is, $p \le m^*$. Now we employ the tools of

Authorized licensed use limited to: Trial User - Warsaw University (Uniwersytet Warszawski). Downloaded on August 09,2023 at 11:58:59 UTC from IEEE Xplore. Restrictions apply.

Section VII-E to translate this sequence to a grammar. We work through the sequence from left-to-right and build a balanced binary grammar as we go. Throughout, we maintain an *active list* of grammar symbols.

Initially, the active list is L_1, which must be a terminal. In general, at the beginning of ith step, the expansion of the active list is the string represented by $L_1 \dots L_i$. Our goal for the step is to augment the grammar and alter the active list so that the expansion of the symbols in the active list is the string represented by $L_1 \dots L_{i+1}$.

If L_{i+1} is a terminal, we can accomplish this goal by simply appending L_{i+1} to the active list. If L_{i+1} is a pair, then it specifies a substring β_i of the expansion of the active list. If β_i is contained in the expansion of a single symbol in the active list, then we use ADDSUBSTRING to create a nonterminal with expansion β_i using $O(\log |\beta_i|)$ rules. This nonterminal is then appended to the active list.

On the other hand, if β_i is not contained in the expansion of a single symbol in the active list, then it is the concatenation of a suffix of $\langle X \rangle$, all of $\langle A_1 \dots A_{t_i} \rangle$, and a prefix of $\langle Y \rangle$, where $XA_1 \dots A_{t_i} Y$ are consecutive symbols in the active list. We then perform the following operations.

1) Construct a nonterminal M with expansion $\langle A_1 \dots A_{t_i} \rangle$ using ADDSEQUENCE. This produces $O(t_i(1+\log |\beta_i|/t_i))$ rules.
2) Replace $A_1 \dots A_{t_i}$ in the active list by the single symbol M.
3) Construct a nonterminal X' with expansion equal to the prefix of β_i in $\langle X \rangle$ using ADDSUBSTRING. Similarly, construct a nonterminal Y' with expansion equal to the suffix of β_i in $\langle Y \rangle$ using ADDSUBSTRING. This produces $O(\log |\beta_i|)$ new rules in total.
4) Create a nonterminal N with expansion $\langle X'MY' \rangle$ using ADDSEQUENCE on X', M, and Y'. This creates $O(\log |\beta_i|)$ new rules. Append N to the end of the active list.

Thus, in total, we add $O(t_i + t_i \log |\beta_i|/t_i + \log |\beta_i|)$ new rules during each step. The total number of rules created is

$$O\left(\sum_{i=1}^{p} t_i + t_i \log |\beta_i|/t_i + \log |\beta_i|\right) = O\left(\sum_{i=1}^{p} t_i + \sum_{i=1}^{p} t_i \log |\beta_i|/t_i + \sum_{i=1}^{p} \log |\beta_i|\right).$$

The first sum is upper-bounded by the total number of symbols inserted into the active list. This is at most two per step (M and N), which gives a bound of $2p$: $\sum_{i=1}^{p} t_i \le 2p$. To upper-bound the second sum, we use the concavity inequality

$$\sum_{i=1}^{p} a_i \log b_i \le \left(\sum_{i=1}^{p} a_i\right) \log\left(\frac{\sum_{i=1}^{p} a_i b_i}{\sum_{i=1}^{p} a_i}\right)$$

and set $a_i = t_i, b_i = |\beta_i|/t_i$ to give

$$\sum_{i=1}^{p} t_i \log \frac{|\beta_i|}{t_i} \le \left(\sum_{i=1}^{p} t_i\right) \log\left(\frac{\sum_{i=1}^{p} |\beta_i|}{\sum_{i=1}^{p} t_i}\right) = O\left(p \log\left(\frac{n}{p}\right)\right).$$

The latter inequality uses the fact that $\sum_{i=1}^{p} |\beta_i| \le n$ and that $\sum_{i=1}^{p} t_i \le 2p$. Note that the function $x \log n/x$ is increasing for x up to n/e, and so this inequality holds only if $2p \le n/e$. This condition is violated only when input string (length n) turns out to be only a small factor $(2e)$ longer than the LZ77 sequence (length p). If we detect this special case, then we can output the trivial grammar $S \to \sigma$ and achieve a constant approximation ratio.

By concavity again, the third sum is upper-bounded by

$$p \log \frac{\sum |\beta_i|}{p} \le p \log \frac{n}{p}.$$

The total grammar size is therefore

$$O\left(p \log \frac{n}{p}\right) = O\left(m^* \log \frac{n}{m^*}\right)$$

where we use the inequality $p \le m^*$ and, again, the observation that $x \log n/x$ is increasing for $x < n/e$. This proves the claim.

G. Grammar-Based Compression Versus LZ77

We have now shown that a grammar of size m can be translated into an LZ77 sequence of length at most m. In the reverse direction, we have shown that an LZ77 sequence of length p can be translated to a grammar of size $O(p \log n/p)$. Furthermore, the latter result is nearly the best possible. Consider strings of the form

$$\sigma = x^{k_1} \mid x^{k_2} \mid \dots \mid x^{k_q}$$

where k_1 is the largest of the k_i. This string can be represented by an LZ77 sequence of length

$$O(q + \log k_1)\colon x\ (1,1)\ (1,2)\ (1,4)\ (1,8) \dots (1, k_i - 2^j) \mid (1,k_2) \mid \dots \mid (1,k_q).$$

Here, j is the largest power of 2 less than k_i. If we set $q = \Theta(\log k_1)$, then the sequence has length $O(\log k_1)$.

On the other hand, Theorem 2 states that the smallest grammar for σ is within a constant factor of the shortest addition chain containing $k_1, \dots, k_q$. Pippinger [37] has shown, via a counting argument, that there exist integers $k_1, \dots, k_q$ such that the shortest addition chain containing them all has length

$$\Omega\left(\log k_1 + q \cdot \frac{\log k_1}{\log \log k_1 + \log q}\right).$$

If we choose $q = \Theta(\log k_1)$ as before, then the above expression boils down to

$$\Omega\left(\frac{\log^2 k_1}{\log \log k_1}\right).$$

Putting this all together, we have a string σ of length $n = O(k_1 \log k_1)$ for which there exists an LZ77 sequence of length $O(\log k_1)$, but for which the smallest grammar has size $\Omega((\log^2 k_1)/(\log \log k_1))$. The ratio between the grammar size and the length of the LZ77 sequence is therefore

$$\Omega\left(\frac{\log k_1}{\log \log k_1}\right) = \Omega\left(\frac{\log n}{\log \log n}\right).$$

Thus, our algorithm for transforming a sequence of LZ77 triples into a grammar is almost optimal.

Authorized licensed use limited to: Trial User - Warsaw University (Uniwersytet Warszawski). Downloaded on August 09,2023 at 11:58:59 UTC from IEEE Xplore. Restrictions apply.

The analysis in this section brings to light the relationship between the best grammar-based compressors and LZ77. One would expect the two to achieve roughly comparable compression performance since the two representations are quite similar. Which approach achieves superior compression (over all cases) in practice depends on many considerations beyond the scope of our theoretical analysis. For example, one must bear in mind that a grammar symbol can be represented by fewer bits than an LZ77 pair. In particular, each LZ77 pair requires about $2 \log n$ bits to encode, although this may be somewhat reduced by representing the integers in each pair with a variable-length code. On the other hand, each grammar symbol can be naively encoded using about $\log m$ bits, which could be as small as $\log \log n$. This can be further improved via an optimized arithmetic encoding as suggested in [4]. Thus, the fact that grammars are can be somewhat larger than LZ77 sequences may be roughly offset by the fact that grammars can also translate into fewer bits. Empirical comparisons in [4] suggest precisely this scenario, but they do not yet seem definitive one way or the other [4], [9], [6], [7], [10], [11], especially in the low-entropy case.

The procedures presented here are not ready for immediate use as practical compression algorithms. The numerous hacks and optimizations needed in practice are lacking. Our algorithms are designed not for practical performance, but for good, *analyzable* performance. In practice, the best grammar-based compression algorithm may yet prove to be a simple scheme like RE-PAIR, which we do not yet know how to analyze.

VIII. FUTURE DIRECTIONS

A. Analysis of Global Algorithms

Our analysis of previously proposed algorithms for the smallest grammar problem leaves a large gap of understanding surrounding the global algorithms, GREEDY, LONGEST MATCH, and RE-PAIR. In each case, we upper-bound the approximation ratio by $O((n/\log n)^{2/3})$ and lower-bound it by some expression that is $o(\log n)$. Elimination of this gap would be significant for several reasons. First, these algorithms are important; they are simple enough to be practical for applications such as compression and DNA entropy estimation. Second, there are natural analogues to these global algorithms for other hierarchically structured problems. Third, all of our lower bounds on the approximation ratio for these algorithms are well below the $\Omega(\log n/\log \log n)$ hardness implied by the reduction from the addition chain problem. Either there exist worse examples for these algorithms or else a tight analysis will yield progress on the addition chain problem.

B. Algebraic Extraction

The need for a better understanding of hierarchical approximation problems beyond the smallest grammar problem is captured in the smallest AND-circuit problem. Consider a digital circuit which has several input signals and several output signals. The function of each output is a specified sum-of-products over the input signals. How many two-input AND gates must the circuit contain to satisfy the specification?

This problem has been studied extensively in the context of automated circuit design. Interestingly, the best known algorithms for this problem are closely analogous to the GREEDY and RE-PAIR algorithms for the smallest grammar problem. (For details on these analogues, see [38], [39], and [40], respectively.) No approximation guarantees are known.

C. String Complexity in Other Natural Models

One motivation for studying the smallest grammar problem was to shed light on a computable and approximable variant of Kolmogorov complexity. This raises a natural follow-on question: can the complexity of a string be approximated in other natural models? For example, the grammar model could be extended to allow a nonterminal to take a parameter. One could then write a rule such as $T(P) \rightarrow PP$, and write the string $xxyzyz$ as $T(x)T(yz)$. Presumably as model power increases, approximability decays to incomputability. Good approximation algorithms for strong string-representation models could be applied wherever the smallest grammar problem has arisen.

ACKNOWLEDGMENT

The authors sincerely thank Yevgeniy Dodis, Martin Farach-Colton, Michael Mitzenmacher, Madhu Sudan, and the reviewers for helpful comments.

REFERENCES

[1] J. Ziv and A. Lempel, "Compression of individual sequences via variable-rate coding," *IEEE Trans. Inf. Theory*, vol. IT-24, no. 5, pp. 530–536, Sep. 1978.

[2] T. A. Welch, "A technique for high-performance data compression," *Computer Mag. Computer Group News of the IEEE Computer Group Soc.*, vol. 17, no. 6, pp. 8–19, 1984.

[3] J. C. Kieffer, E. H. Yang, G. J. Nelson, and P. Cosman, "Universal lossless compression via multilevel pattern matching," *IEEE Trans. Inf. Theory*, vol. 46, no. 5, pp. 1227–1245, Jul. 2000.

[4] E. H. Yang and J. C. Kieffer, "Efficient universal lossless data compression algorithms based on a greedy sequential grammar transform—Part one: Without context models," *IEEE Trans. Inf. Theory*, vol. 46, no. 3, pp. 755–777, May 2000.

[5] J. C. Kieffer and E. H. Yang, "Grammar based codes: A new class of universal lossless source codes," *IEEE Trans. Inf. Theory*, vol. 46, no. 3, pp. 737–754, May 2000.

[6] C. G. Nevill-Manning, "Inferring sequential structure," Ph.D. dissertation, University of Waikato, Hamilton, New Zealand, 1996.

[7] N. J. Larsson and A. Moffat, "Offline dictionary-based compression," *Proc. IEEE*, vol. 88, no. 11, pp. 1722–1732, Nov. 2000.

[8] J. K. Lanctot, M. Li, and E. H. Yang, "Estimating DNA sequence entropy," in *Proc. Symp. Discrete Algorithms*, San Francisco, CA, Jan. 2000, pp. 409–418.

[9] C. G. de Marcken, "Unsupervised language acquisition," Ph.D. dissertation, MIT, Cambridge, MA, 1996.

[10] A. Apostolico and S. Lonardi, "Off-line compression by greedy textual substitution," *Proc. IEEE*, vol. 88, no. 11, pp. 1733–1744, Nov. 2000.

[11] ——, "Some theory and practice of greedy off-line textual substitution," in *Proc. IEEE Data Compression Conf., DCC*, Snowbird, UT, Mar. 1998, pp. 119–128.

[12] A. N. Kolmogorov, "Three approaches to the quantitative definition of information," *Probl. Inf. Transm.*, pp. 1–7, 1965.

[13] C. G. Nevill-Manning and I. H. Witten, "Identifying hierarchical structure in sequences: A linear-time algorithm," *J. Artificial Intell.*, vol. 7, pp. 67–82, 1997.

[14] T. Kida, Y. Shibata, M. Takeda, A. Shinohara, and S. Arikawa, "A unifying framework for compressed pattern matching," in *Proc. 6th Int. Symp. String Processing and Information Retrieval*, Cancun, Mexico, 1999, pp. 89–96.

Authorized licensed use limited to: Trial User - Warsaw University (Uniwersytet Warszawski). Downloaded on August 09,2023 at 11:58:59 UTC from IEEE Xplore. Restrictions apply.

[15] J. C. Kieffer and E. H. Yang, "Sequential codes, lossless compression of individual sequences, and kolmogorov complexity," *IEEE Trans. Inf. Theory*, vol. 42, no. 1, pp. 29–39, Jan. 1996.

[16] G. Nelson, J. C. Kieffer, and P. C. Cosman, "An interesting hierarchical lossless data compression algorithm," in *Proc. IEEE Information Theory Society Workshop*, Rydzyna, Poland, Jun. 1995. Invited Presentation.

[17] R. E. Bryant, "Graph-based algorithms for Boolean function manipulation," *IEEE Trans. Comput.*, vol. C-35, no. 8, pp. 677–691, Aug. 1986.

[18] J. C. Kieffer, P. Flajolet, and E.-H. Yang, "Data compression via binary decision diagrams," in *IEEE Int. Symp. Information Theory*, vol. 46, Jun. 2000, p. 296.

[19] C.-H. Lai and T.-F. Chen, "Compressing inverted files in scalable information systems by binary decision diagram encoding," in *Proc. 2001 Conf. Supercomputing*, Denver, CO, Nov. 2001, p. 60.

[20] A. Lempel and J. Ziv, "On the complexity of finite sequences," *IEEE Trans. Inf. Theory*, vol. IT-23, no. 1, pp. 75–81, Jan. 1976.

[21] J. Ziv and A. Lempel, "A universal algorithm for sequential data compression," *IEEE Trans. Inf. Theory*, vol. IT-23, no. 3, pp. 337–343, May 1977.

[22] J. A. Storer, "Data compression: Methods and complexity," Ph.D. dissertation, Princeton Univ., Princeton, NJ, 1978.

[23] J. A. Storer and T. G. Szymanski, "Data compression via textual substitution," *J. ACM*, vol. 29, no. 4, pp. 928–951, Oct. 1982.

[24] J. A. Storer, *Data Compression: Methods and Theory*. Rockville, MD: Computer Science, 1988.

[25] A. Apostolico and S. Lonardi, "Compression of biological sequences by greedy off-line textual substitution," in *Proc. IEEE Data Compression Conf., DCC*, Snowbird, UT, Mar. 2000, pp. 143–152.

[26] Y. Shibata, T. Kida, S. Fukamachi, M. Takeda, A. Shinohara, T. Shinohara, and S. Arikawa, "Byte Pair Encoding: A Text Compression Scheme That Accelerates Pattern Matching," Dept., Kyushu Univ., Kyushu, Japan, Tech. Rep. DOI-TR-CS-161, Apr. 1999.

[27] S. R. Kosaraju and G. Manzini, "Compression of low entropy strings with Lempel-Ziv algorithms," *SIAM J. Comput.*, vol. 29, no. 3, pp. 893–911, 2000.

[28] P. Berman and M. Karpinski, "On Some Tighter Inapproximability Results, Further Improvements," Electronic Colloquium on Computational Complexity, Tech. Rep. TR98-065, 1998.

[29] D. E. Knuth, *Seminumerical Algorithms*. Reading, MA: Addison-Wesley, 1981.

[30] E. G. Thurber, "Efficient generation of minimal length addition chains," *SIAM J. Comput.*, vol. 28, no. 4, pp. 1247–1263, 1999.

[31] P. Erdös, "Remarks on number theory III," *ACTA Arithmetica*, vol. VI, pp. 77–81, 1960.

[32] P. Downey, B. Leong, and R. Sethi, "Computing sequences with addition chains," *SIAM J. Comput.*, vol. 10, no. 3, pp. 638–646, Aug. 1981.

[33] A. C.-C. Yao, "On the evaluation of powers," *SIAM J. Comput.*, vol. 5, no. 1, pp. 100–103, Mar. 1976.

[34] P. Gage, "A new algorithm for data compression," *The C Users J.*, vol. 12, no. 2, Feb. 1994.

[35] A. Blum, T. Jiang, M. Li, J. Tromp, and M. Yannakakis, "Linear approximation of shortest superstrings," in *Proc. Symp. Theory of Computing*, New Orleans, LA, May 1991, pp. 328–336.

[36] C. G. Nevill-Manning and I. H. Witten, "Compression and explanation using hierarchical grammars," *Comput. J.*, vol. 40, no. 2/3, pp. 103–116, 1997.

[37] N. Pippenger, "On the evaluation of powers and monomials," *SIAM J. Comput.*, vol. 9, no. 2, pp. 230–250, May 1980.

[38] R. K. Brayton and C. McMullen, "The decomposition and factorization of boolean expressions," in *Proc. Int. Symp. Circuits and Systems*, Rome, Italy, 1982, pp. 49–54.

[39] R. K. Brayon, R. L. Rudell, A. L. Sangiovanni-Vincentelli, and A. R. Wang, "Multi-level logic optimization and the rectangle covering problem," in *Proc. Int. Conf. Computer Aided Design*, San Francisco, CA, Nov. 1987, pp. 66–69.

[40] J. Rajski and J. Vasudevamurthy, "The testability-preserving concurrent decomposition and factorization of boolean expressions," *IEEE Trans. Computer-Aided Design*, vol. 11, no. 6, pp. 778–793, Jun. 1992.

Authorized licensed use limited to: Trial User - Warsaw University (Uniwersytet Warszawski). Downloaded on August 09,2023 at 11:58:59 UTC from IEEE Xplore. Restrictions apply.