Grammar Compression, LZ-encodings and String Algorithms with Implicit Input

Wojciech Rytter *

Instytut Informatyki, Warsaw University, Poland,
and Department of Computer Science, NJIT, USA
Email: rytter@oak.njit.edu

Abstract. We survey several complexity issues related to algorithmic problems on words given in an *implicit* way: by a *grammar*, LZ*-encoding* or as a minimal solution of a *word equation*. We discuss the relation between two implicit representations, the role of word compression in solvability of word equations and compressed language recognition problems. The grammar compression is more convenient than LZ-encoding, its size differs from that of LZ-encoding by at most logarithmic factor, the constructive proof is based on the concept similar to balanced trees.

1 Introduction

Algorithmics on compressed objects is a recently developed area of theoretical computer science. Its is motivated by the increase in the volume of data and the need to store and transmit masses of information in *compressed* form. The compressed information has to be quickly accessed and processed without explicit decompression. The main problem is how to deal efficiently with strings given implicitly.

We discuss three types of implicit representation: a context free grammar generating a single string, a Lempel-Ziv encoding and word equations describing a string as a minimal solution. The last type representation is the most complex, the best upper bound on the size of minimal solution is doubly exponential while it is believed that it is only singly exponential. The implicit representations of strings is the main tool in the best known algorithms for testing solvability of word equations.

Lempel-Ziv encodings.

Intuitively, LZ algorithm compresses the input word because it is able to discover some repeated subwords, see [8]. The Lempel-Ziv code defines a natural factorization of the encoded word into subwords which correspond to intervals in the code. The subwords are called *factors*. Assume that Σ is an underlying alphabet and let w be a string over Σ. The LZ-factorization of w is given by a decomposition: $w = f_1 \cdot f_2 \cdot \cdot f_k$, where $f_1 = w[1]$ and for each $2 \leq i \leq k$, f_i

* Supported by the grants ITR-CCR-0313219 and KBN 4T11C04425

is the longest prefix of $f_i \dots f_k$ which occurs in $f_1 \dots f_{i-1}$ or a single symbol if there is no such nonempty prefix. We can identify each f_i with an interval $[p, q]$, such that $f_i = w[p \dots q]$ and $q \le |f_1 \dots f_{i-1}|$. We identify LZ-factorization with $LZ(w)$. Its size if the number of factors.

Example 1. The LZ-factorization of the 7-th Fibonacci word Fib_7 is given by:

$$abaababaabaab \;=\; f_1\ f_2\ f_3\ f_4\ f_5\ f_6 \;=\; a\ b\ a\ aba\ baaba\ ab$$

Grammar compression.

Text compression based on context free grammars, or equivalently, on straight-line programs, has attracted much attention, see [5, 20, 24–26, 38]. The grammars give a more structured type of compression and are more convenient for example in compressed pattern-matching, see [38]. In a grammar-based compression a single text w of length n is generated by a context-free grammar G. Computing exact size of the minimal grammar-based compression is known to be NP-complete. For simplicity assume that the grammars are in Chomsky normal form. The size of the grammar G, denoted by $|G|$, is the number of productions (rules), or equivalently the number of nonterminals of a grammar G in Chomsky normal form. Grammar compression is essentially equivalent to straight-line programs. A *grammar* (*straight-line program*) is a sequence:

$$X_1 = expr_1;\ X_2 = expr_2; \dots ;\ X_m = expr_m,$$

where X_i are nonterminals and $expr_i$ is a single terminal symbol, or $expr_i$ is of a form $X_j \cdot X_k$, for some $j, k < i$, where $\cdot$ denotes the concatenation of X_j and X_k. For each nonterminal X_i, denote by $val(X_i)$ the value of X_i, it is the string described by X_i. The string described by the whole straight-line program is $val(X_m)$. The size of the straight-line program is m.

Example 2.
Let us consider the following grammar G_7 which describes the 7*th Fibonacci word* $Fib_7 = abaababaabaab$. We have $|G_7| = 7$. This is the smallest size grammar in Chomsky normal form for Fib_7.

$$X_7 = X_6 \cdot X_5;\ \ X_6 = X_5 \cdot X_4;\ \ X_5 = X_4 \cdot X_3;$$
$$X_4 = X_3 \cdot X_2;\ \ X_3 = X_2 \cdot X_1;\ \ X_2 = a;\ \ X_1 = b;$$

Word equations.

The problem of solving word equations is not well understood. Word equations can be used to define various properties of strings, e.g. general versions of pattern-matching with variables. Instead of dealing with very long solutions we can deal with their Lempel-Ziv encodings. Each minimal solution of a word equation is highly compressible (exponentially compressible for long solutions) in terms of Lempel-Ziv encoding. The best known algorithm for general word equations works in P-$SPACE$. The polynomial space complexity is possible due to implicit representation of huge strings.

Compressed membership problems for formal languages.

The problem consists in checking if an input word w is in a given language L, when we are only given a compressed representation of w. We present several results related to language recognition problems for compressed texts. These problems are solvable in polynomial time for uncompressed words and some of them become NP-hard for compressed words. The complexity depends on the type and description of the language L. In particular the membership problem is in polynomial-time for regular expressions. It is P-TIME complete for a fixed regular language. However it is NP-hard for semi-extended regular expressions and P-SPACE complete for context-free languages. The membership problem is NP-complete for unary regular expressions with compressed constants. Also for unary languages compressed recognition of context-free languages is NP-complete.

2 Relation Between Minimal Grammar Compression and LZ-encodings.

The problem of finding the smallest size grammar (or equivalently, straight line program) generating a given text is NP-complete. If A is a nonterminal of a grammar G then we sometimes identify A with the grammar G with the starting nonterminal replaced by A, all useless unreachable nonterminals being removed. In the parse tree for a grammar with the starting nonterminal A we can also sometimes informally identify A with the root of the parse tree.

For a grammar G generating w we define the parse-tree $Tree(G)$ of w as a derivation tree of w, in this tree we identify (conceptually) terminal symbols with their parents, in this way every internal node has exactly two sons. Define the partial parse-tree, denoted $PTree(G)$ as a maximal subtree of $Tree(G)$ such that for each internal node there is no node to the left having the same label. We define also the grammar factorization, denoted by G-factorization, of w, as a sequence of subwords generated by consecutive bottom nonterminals of $PTree(G)$. Alternatively we can define G-factorization as follows: w is scanned from left to right, each time taking as next G-factor the longest unscanned prefix which is generated by a single nonterminal which has already occurred to the left or a single letter if there is no such nonterminal. The factors of LZ- and G-factorizations are called LZ-factors and G-factors, respectively.

Example 3. The G_7-factorization for the 7-th Fibonacci strings is:

$$g_1\ g_2\ g_3\ g_4\ g_5\ g_6\ =\ a\ b\ a\ ab\ aba\ abaab$$

It can be shown that the number of factors in LZ-factorizations is not larger than the number of G-factors.

Theorem 1.
For each string w and its grammar-based compression G $|LZ(w)| \leq |G|$.

AVL-grammars.

AVL-grammars correspond naturally to AVL-trees. The first use of a different type balanced grammars has appeared in [17]. AVL-trees are usually used in the context of binary search trees, here we use them in the context of storing in the leaves the consecutive symbols of the input string w. The basic operation is the concatenation of sequences of leaves of two trees. We use the standard AVL-trees, for each node v the balance of v, denoted $bal(v)$ is the difference between the height of the left and right subtrees of the subtree of T rooted at v. T is *AVL-balanced* iff $|bal(v)| \leq 1$ for each node v. We say that a grammar G is *AVL-balanced* if $Tree(G)$ is *AVL-balanced*. Denote by $height(G)$ the height of $Tree(G)$ and by $height(A)$ the height of the parse tree with the root labelled by a nonterminal A. The following fact is a consequence of a similar fact for AVL-trees, see [22].

Lemma 1. *If the grammar G is AVL-balanced then $height(G) = O(\log n)$.*

In case of *AVL-balanced* grammars in each nonterminal A additional information about the balance of A is kept: $bal(A)$ is the balance of the node corresponding to A in the tree $Tree(G)$. We do not define the balance of nodes corresponding to terminal symbols, they are identified with their fathers: nonterminals generating single symbols. Such nonterminals are leaves of $Tree(G)$, for each such nonterminal B we define $bal(B) = 0$.

Example 4. The grammar G_7 for the 7th Fibonacci word is *AVL-balanced*.

Lemma 2. *Assume A, B are two nonterminals of AVL-balanced grammars. Then we can construct in $O(|height(A) - height(B)|)$ time a AVL-balanced grammar $G = Concat(A, B)$, where $val(G) = val(A) \cdot val(B)$, by adding only $O(|height(A) - height(B)|)$ nonterminals.*

Assume we have an LZ-factorization $f_1 f_2 \dots f_k$ of w. We convert it into a grammar whose size increases by a logarithmic factor. Assume we have LZ-factorization $w = f_1 f_2 \dots f_k$ and we have already constructed *good* (*AVL-balanced* and of size $O(i \cdot \log n)$) grammar G for the prefix $f_1 f_2 \dots f_{i-1}$. If f_i is a terminal symbol generated by a nonterminal A then we set $G := Concat(G, A)$. Otherwise we locate the segment corresponding to f_i in the prefix $f_1 f_2 \dots f_{i-1}$. Due to the fact that G is balanced we can find a logarithmic number of nonterminals $S_1,\ S_2, \dots S_{t(i)}$ of G such that $f_i = val(S_1) \cdot val(S_2) \cdot \dots val(S_{t(i)})$. The sequence $S_1, S_2, \dots S_{t(i)}$ is called the *grammar decomposition* of the factor f_i.

We concatenate the parts of the grammar corresponding to this nonterminals with G, using the operation $Concat$ mentioned in Lemma 2. Assume the first $|\Sigma|$ nonterminals corresponds to letters of the alphabet, so they exist at the beginning. We initialize G to the grammar generating the first symbol of w and containing all nonterminals for terminal symbols, they don't need to be initially *connected* to the string symbol. If LZ-factorization is too large (exceeds $n/\log n$) then we neglect it and write a trivial grammar of size n generating a given string. Otherwise we have only $k \leq n \cdot \log n$ factors, they are processed from left to right. We perform the algorithm *Construct-Grammar*.

ALGORITHM *Construct-Grammar*; {for string w of size n}
we are given LZ factorization $f_1 f_2 f_3 \ldots f_k$ of w
if $k > n/\log(n)$ **then return** trivial $O(n)$ size grammar
else for $i = 1$ **to** k **do**
(1) Let $S_1, S_2, \ldots, S_{t(i)}$ be grammar decomposition of f_i;
(2) $H := Concat(S_1, S_2, \ldots, S_{t(i)})$;
(3) $G := Concat(G, H)$;

Due to Lemma 2 we have $t(i) = O(\log n)$, so the number of two-arguments concatenations needed to implement single step (2) is $O(\log n)$, each of them adding $O(\log n)$ nonterminals. Steps (1) and (3) can be done in $O(\log n)$ time, since the height of the grammar is logarithmic. Hence the algorithm gives $O(\log^2(n)$-ratio approximation. At the cost of slightly more complicated implementation of step (2) $\log^2 n$-ratio can be improved to a $\log n$-ratio approximation. The key observation is that the sequence of heights of subtrees corresponding to segments S_i of next LZ-factor is *bitonic*. We can split this sequence into two subsequences: height-nondecreasing sequence $R_1, R_2, \ldots R_k$, called *right-sided*, and height-nonincreasing sequence $L_1, L_2, \ldots L_r$, called *left-sided*.

Lemma 3. *Assume $R_1, R_2, \ldots R_k$ is a right-sided sequence, and G_i is the AVL-grammar which results by concatenating $R_1, R_2, \ldots R_i$ from left-to-right. Then* $|height(R_i) - height(G_{i-1})| \ \leq\ \max\ \{(height(R_i) - height(R_{i-1}),\ 1\}$

Theorem 2. *We can construct in a $O(n \log |\Sigma|)$ time a $O(\log n)$-ratio approximation of a minimal grammar-based compression.*
Given LZ-factorization of length k we can construct a corresponding grammar of size $O(k \log n)$ in time $O(k \log n)$.

Proof. The next factor f_i is decomposed into segments $S_1, S_2, \ldots S_{t(i)}$. It is enough to show that we can create in $O(\log n)$ time an AVL-grammar for the concatenation of $S_1, S_2, \ldots, S_{t(i)}$ by adding only $O(\log n)$ nonterminals and productions to G, assuming that the grammars for $S_1, S_2, \ldots, S_{t(i)}$ are available.

The sequence $(S_1, S_2, \ldots, S_{t(i)}$ consists of a right-sided sequence and left-sided sequence. The grammars $H',\ H''$ corresponding to these sequences are computed (by adding logarithmically many nonterminals to G), due to Lemma 3. Then $H',\ H''$ are concatenated. Assume $R_1, R_2, \ldots R_k$ are right-sided subtrees. Then the total work and number of extra nonterminals needed to concatenate $R_1, R_2, \ldots R_k$ can be estimated as follows:

$$\sum_{i=2}^{k} |height(R_i) - height(G_{i-1})| \ \leq\ \sum_{i=2}^{k} \max\ \{height(R_i) - height(R_{i-1}),\ 1\}$$

$$\leq\ \sum_{i=2}^{k} (height(R_i) - height(R_{i-1})) + \sum_{i=2}^{k}\ 1\ \ \leq height(R_k) + k\ =\ O(\log n).$$

The same applies to the left-sided sequence in a symmetric way. Altogether processing each factor f_i enlarges the grammar by an $O(\log n)$ additive factor and needs $O(\log n)$ time. To get $\log n$-ratio we consider only the case when the number k of factors is $O(n/\log n)$. LZ-factorization is computed in $O(n \log |\Sigma|)$ time using suffix trees, ($O(n)$ time for integer alphabets, see [19,11])).

There is possible a rather cosmetic improvement of the approximation ratio. Let g be the size of the minimal grammar-based compression and assume we have a greedy LZ-factorization of a string w of size n into s factors, the number s is also a lower bound on g. The improvement is a direct application of a method from the paper on compressed matching of Farach and Thorup [10], (In their notation $n = U,\ g = n$). In [10] they improved a starting factor $\log n$ to $\log(n/g)$ by introducing new cut-points and refining factorization. Exactly in the same way $\log n$ can be improved to get $\log(n/g)$.

Theorem 3. [6,37]
We can construct in polynomial time $O(\log(n/g))$-ratio approximation of a minimal grammar compression, where g is the size of the minimal grammar based compression of a given string of length n.

3 String Compression and Word Equations

Word equations are used to describe properties and relations of words, e.g. pattern-matching with variables, imprimitiveness, periodicity, conjugation, [18].

Let Σ be an alphabet of constants and Θ be an alphabet of variables. We assume that these alphabets are disjoint. A word equation E is a pair of words $(u,v) \in (\Sigma \cup \Theta)^* \times (\Sigma \cup \Theta)^*$ usually denoted by $u = v$. The *size* of an equation is the sum of lengths of u and v. A *solution* of a word equation $u = v$ is a morphism $h : (\Sigma \cup \Theta)^* \to \Sigma^*$ such that $h(a) = a$, for $a \in \Sigma$, and $h(u) = h(v)$. For example assume we have the equation

$$abx_1x_2x_2x_3x_3x_4x_4x_5 = x_1x_2x_3x_4x_5x_6,$$

and the length of x_i's are consecutive Fibonacci numbers. Then the solution is $h(x_i) = Fib_i$.

It is known that the solvability problem for word equations is in $P\text{-}SPACE$ and is NP-hard, even if we consider (short) solutions with the length bounded by a linear function and the right side of equations contains no variables, see [4]. The main open problem is to close the gap between NP and $P\text{-}SPACE$, and to show the following

Conjecture A: the problem of solving word equations is in NP.

Assume n is the size of the equation and N is the minimal length of the solution (if one exists). It is generally believed that another conjecture is true (at least no counterexample is known):

Conjecture B: N is at most singly exponential w.r.t. n.

A motivation to consider compressed solutions follows from the following fact.

Lemma 4. [34]
If we have grammar-encoded values of the variables then we can verify the word equation in polynomial time with respect to the size of the equation and the total size of the encodings.

Assume $h(u) = h(v) = \mathcal{T}$ is a solution of a given word equation E. A *cut* in $\mathcal{T}$ is a border of a variable or a constant in $\mathcal{T}$. We say that a subword w of $\mathcal{T}$ *overlaps* a cut γ iff an occurrence of w extends to the left and right of γ or γ is a border of an occurrence.

Lemma 5 (overlap lemma). [36]
Assume $\mathcal{T}$ is the minimal length solution of the equation E. Then each subword of $\mathcal{T}$ has an occurrence which overlaps at least one cut in $\mathcal{T}$.

The overlap lemma is crucial in proving the following fact.

Theorem 4. [36]
Assume N is the size of minimal solution of a word equation of size n. Then each solution of size N can be LZ-compressed to a string of size $O(n^2 \log^2(N)(\log n + \log\log N))$.

As a direct consequence we have:

Corollary 1. Conjecture B implies conjecture A.

Proof.
If N is exponential then the compressed version of the solution is of a polynomial size. The algorithm below solves the problem in nondeterministic polynomial time. The first step works in nondeterministic polynomial time, the second one works in deterministic polynomial time due to Lemma 4.

> **ALGORITHM** *Solving_by_LZ-Encoding* ;
>
> guess LZ-encoded solution of size
> $O(n^2 \log^2 N(\log n + \log\log N))$;
> verify its correctness using the polynomial
> time deterministic algorithm from Lemma 4.

Using more complicated algorithm it can be shown the following:

Theorem 5. *Assume the length of all variables are given in binary by a function f. Then we can test solvability in polynomial time, and produce polynomial-size compression of the lexicographically first solution (if there is any).*

The periodicity index for a given string x is the maximal k such that u^k is a subword of x, for a nonempty u.

Example 5.
For example index of periodicity of *abbababababababbaba* is 5 since we can write:

$$abbababababababbaba = \; abb(ab)^5baba;$$

Denote by $index\_per(n)$ the maximal index of periodicity of minimal length solutions of word equations of length n.

Theorem 6 (periodicity index lemma). [23]
$index\_per \leq 2^{cn}$ for a constant c.

It has been shown by W. Plandowski that the solvability of word equations is in P-SPACE, this algorithm is the milestone achievement is this area. The algorithm consists of nondeterministically finding a (compressed) compressed syntactic derivation of the equation. All equations in the derivation have lengths bounded by $p(n)$, where $p(n)$ is a fixed polynomial (the same for all equations) and n is the size of the original equation.

Assume Γ is the set of additional variables called pseudo-constants, and Σ is the set of original constants. Let $\mathcal{X}$ be the set of original variables, assume $\mathcal{X} \cap \Gamma \; = \; \emptyset$.

An exponential expression is a compressed representation of a word in terms of concatenation and exponentiation, e.g.

$$a^{237}bac^{1024}cdb^{23}$$

with the main invariant: all exponents are singly exponential w.r.t. n, hence can be stored in P-SPACE.

Each step in the compressed syntactic derivation should be accompanied by all possible reductions of the exponential expressions, to guarantee that their height is at most one. This can be done nondeterministically, guessing the final form and testing if the initial expression and the required one are equivalent. The *compressed syntactic derivation* consists in performing locally in one step one of the following syntactic operations :

1. Replace **each** occurrence of a selected pseudo-constant by an exponential expressions of size $O(n^3)$ over the alphabet $\Gamma \cup \Sigma$;
2. Replace a subword α over the alphabet $\Gamma \cup \Sigma$ in the actual equation by a variable $X \in \mathcal{X}$. For the same variable α should be the same in one iteration.
3. Replace a fragment $X \cdot R$, where ($X \in \mathcal{X}$ and $R \in (\Gamma \cup \Sigma)^*$), by X, for an original variable X, this should be done for **all** occurrences of X in the actual equation

We start with $w = w$ and end with the original equation preserving the invariant: the actual equation is solvable We show a *compressed syntactic derivation* of the following equation (the variables are written with capital letters)

$$Y\ Y\ X\ b\ a\ a\ a\ a\ Z \ = \ Z\ Y\ Y\ X\ a\ a\ a\ a\ b$$

The compressed syntactic derivation provides a compressed proof that this equation is solvable.

$$w \ = \ w \stackrel{w\Rightarrow uvuuv}{\longrightarrow} uvuuv \ = \ uvuuv \stackrel{Z\Leftarrow uv}{\longrightarrow} uvuZ \ = \ Zuuv$$

$$\stackrel{v\Rightarrow ce}{\longrightarrow} uceuZ \ = \ Zuuce \stackrel{u\Rightarrow ccd}{\longrightarrow} ccdceccdZ \ = \ Zccdccdce \stackrel{Y\Leftarrow c,\ X\Leftarrow c}{\longrightarrow}$$

$$YcdYecXdZ = ZYcdYcdXe \stackrel{c\Rightarrow ef}{\longrightarrow} YefdYeefXdZ = ZYefdYefdXe \stackrel{Y\Leftarrow Ye}{\longrightarrow}$$

$$YfdYefXdZ = ZYfdYfdXe \stackrel{e\Rightarrow fb}{\longrightarrow} YfdYfbfXdZ = ZYfdYfdXfb$$

$$\stackrel{Y\Leftarrow Yf}{\longrightarrow} YdYbfXdZ = ZYdYdXfb$$

$$\stackrel{d\Rightarrow bf}{\longrightarrow} YbfYbfXbfZ \ = \ ZYbfYbfXfb \stackrel{Y:=\Leftarrow Ybf}{\longrightarrow} YYXbfZ \ = \ ZYYXfb$$

$$\stackrel{f\Rightarrow a^4}{\longrightarrow} YYXbaaaaZ \ = \ ZYYXaaaab$$

Another possibility to check solvability of the equation is to guess the following values for the variables (then test the example word equation).

$$x \ = \ aaaabaaaa; \ \ y \ = \ aaaabaaaaaaaabaaaabaaaa$$
$$z \ = \ aaaabaaaaaaaabaaaabaaaaaaaabaaaaaaaab$$

After substituting these values into the equation the length of each of its sides becomes 97.
Using the overlap-lemma and the periodicty-index lemma the following theorem has been shown. Observe that the only nontrivial part here is the "$\Rightarrow$" implication in point (1).

Theorem 7. [33]
(1) *The word equation has a solution iff it has a syntactic derivation;*
(2) *The solvability problem for word equations is in P-SPACE.*

Let A be a deterministic finite automaton. The pseudo-constants and variables which appear in the syntactic derivation can be augmented by additional information: transition tables which say to which state we move after reading a subword (consisting of final constants) corresponding to the pseudo-constant or the actual value of the variable after starting in each possible state. In this way it can be shown the following:

Theorem 8. *Solvability of word equations with regular constraints (values of variables should be in given regular sets) is P-$SPACE$ complete.*

4 Membership of Compressed Strings in Formal Languages.

For a formal language L and a compressed word x, given implicitly by a grammar, we are to check if $x \in L$. The compressed string matching can be treated as a language recognition problem:

check if $w \in \{x\#y \ : \ x$ is a subword of y, and $x,\ y$ do not contain $\#\}$.

Compressed recognition problems for regular expressions.

We consider three classes of regular expressions as descriptions of regular languages:

1. (standard) regular expressions (using uncompressed constants and operations $\cup,\ ^*, \cdot$);
2. regular expressions with compressed constants (constants given in compressed forms);
3. semi-extended regular expressions (using additionally the operator $\cap$ and only uncompressed constants)

The size of the expression is a part of the input size.

Theorem 9.
(a) *We can decide for compressed words the membership in a language described by given regular expression W in $O(n \cdot m^3)$ time, where $m = |W|$.*
(b) *We can decide for compressed words the membership in a language described by given deterministic automaton M in $O(n \cdot m)$ time, where m is the number of states of M.*

The following problem is useful to show NP-hardness of several compressed recognition problems.

SUBSET SUM problem:
Input instance: Finite set $A = \{a_1, a_2, \ldots, a_n\}$ of integers and an integer K. The size of the input is the number of bits needed for the binary representation of numbers in A and K.
Question: Is there a subset $A' \subseteq A$ such that the sum of the elements in A' is exactly K?

Lemma 6. [12] *The problem SUBSET SUM is NP-complete.*

We show an application of the SUBSET SUM problem in the proof of the following fact.

Theorem 10. [35] *The problem of checking membership of a compressed unary word in a language described by a* star-free *regular expression with compressed constants is NP-complete.*

Proof.
The proof of NP-hardness is a reduction from the SUBSET SUM problem. We can construct easily a straight-line program such that $value(X_i) = d^{a_i}$ and $w = d^K$. Then the SUBSET SUM problem is reduced to the membership:

$$w \in (value(X_1) \cup \varepsilon) \cdot (value(X_2) \cup \varepsilon) \cdots (value(X_n) \cup \varepsilon).$$

The empty string ε can be easily eliminated in the following way. We replace each ε by a single symbol d and each number a_i by $(a_i + 1)$. Then we check whether d^{n+K} is generated by the obtained expression.

The problem is in NP since expressions are *star-free*. We can construct an equivalent nondeterministic finite automaton A and guess an accepting path. All paths are of polynomial length due to *star-free* condition. We can check in polynomial time if concatenation of constants on the path equals an input text P. This completes the proof.

Theorem 11. [35] *The problem of checking membership of a compressed unary word in a language described by a given regular expression with compressed constants is in NP.*

Theorem 12. [35] *The problem of checking membership of a compressed word in a language described by a semi-extended regular expression is NP-hard.*

Recently it has been shown that the membership problem for a fixed regular language is comlete in the class of deterministic polynomial time computations.

Theorem 13. [28] *There is a fixed finite automaton for which the compressed membership problem is P-COMPLETE.*

Compressed membership problem for context-free languages.

The compressed memebrship problem is more difficult than that for regular languages, though in the uncompressed setting both can be soved in deterministic polynomial time.

Theorem 14. [35] *The problem of checking membership of a compressed unary word in a given cfl is NP-complete.*

The compressed context-free memebership problem for general alphabets has surprisingly high complexity status.

Theorem 15. [28]
The compressed membership problem for context free languages is P-SPACE complete.

References

1. A. Amir, G. Benson and M. Farach, *Let sleeping files lie: pattern-matching in Z-compressed files,* in *SODA'94*.
2. A. Amir, G. Benson, *Efficient two dimensional compressed matching, Proc. of the 2nd IEEE Data Compression Conference* 279-288 (1992).
3. A. Amir, G. Benson and M. Farach, *Optimal two-dimensional compressed matching,* in *ICALP'94* pp.215-225.
4. Angluin D., Finding patterns common to a set of strings, *J.C.S.S.*, **21(1)**, 46-62, 1980.
5. A. Apostolico, S. Leonardi, Some theory and practice of greedy off-line textual substitution, DCC 1998, pp. 119-128
6. Moses Charikar, Eric Lehman, Ding Liu, Rina Panigrahy, Manoj Prabhakaran, April Rasala, Amit Sahai and Abhi Shelat, Approximating The Smallest Grammar: Kolmogorov Complexity in Natural Models, STOC 2002
7. Choffrut, C., and Karhumäki, J., Combinatorics of words, *in* G.Rozenberg and A.Salomaa (eds), *Handbook of Formal Languages*, Springer, 1997.
8. M. Crochemore, W. Rytter, *Jewels of stringology - Text algorithms*, World Scientific 2003
9. Eyono Obono, S., Goralcik, P., and Maksimenko, M., Efficient solving of the word equations in one variable, *in Proc. MFCS'94*, LNCS 841, Springer Verlag, 336-341, 1994.
10. Farah, M., Thorup M., String matching in Lempel-Ziv compressed strings, *STOC'95*, 703-712, 1995.
11. Farach, M., Optimal suffix tree construction with large alphabets, FOCS 1997.
12. M.R. Garey and D.S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness,* Freeman, New York (1979).
13. L. Gąsieniec, M. Karpiński, W. Plandowski and W. Rytter, *Efficient algorithms for compressed strings.* in proceedings of the SWAT'96, LNCS 1097, 392-403, 1996.
14. L. Gąsieniec, M. Karpiński, W. Plandowski and W. Rytter, *Randomized Efficient Algorithms for Compressed Strings: the finger-print approach.* in proceedings of the CPM'96, LNCS 1075, 39-49, 1996.
15. L.Gasieniec, W. Rytter, Almost optimal fully compressed LZW-matching, in *Data Compression Conference*, IEEE Computer Society 1999
16. L.Gasieniec, A.Gibbons, W. Rytter, The parallel complexity of pattern-searching in highly compressed texts, in *MFCS* 1999
17. M. Hirao, A. Shinohara, M. Takeda, S. Arikawa, Faster fully compressed pattern matching algorithm for balanced straight-line programs", Proc. of 7th International Symposium on String Processing and Information Retrieval (SPIRE2000), pp. 132-138. IEEE Computer Society, September 2000
18. Karhumäki J., Mignosi F., Plandowski W., The expressibility of languages and relations by word equations, *in ICALP'97*, LNCS 1256, 98-109, 1997.
19. J. Karkkainen, P. Sanders, Simple linear work suffix array construction, ICALP 2003
20. M. Karpinski, W. Rytter and A. Shinohara, *Pattern-matching for strings with short description,* in *Combinatorial Pattern Matching*, 1995.
21. T. Kida, Y. Shibara, M. Takeda, A. Shinohara, S. Arikawa, A unifying framework for compressed pattern matching, SPIRE'99
22. D. Knuth, *The Art of Computing, Vol. II: Seminumerical Algorithms. Second edition.* Addison-Wesley (1981).

23. Koscielski, A., and Pacholski, L., Complexity of Makanin's algorithm, *J. ACM* **43**(4), 670-684, 1996.
24. J. Kieffer, E. Yang, Grammar-based codes: a new class of universal lossless source codes, IEEE Trans. on Inf. Theory 46 (2000) pp. 737-754
25. J. K. Lanctot, Ming Li, En-hui Yang, Estimating DNA Sequence Entropy, SODA 2000
26. E. Lehman, A. Shelat, Approximation algorithms for grammar-based compression, SODA 2002
27. A. Lempel, J. Ziv, On the complexity of finite sequences, *IEEE Trans. on Inf. Theory*, 22, 75-81, 1976.
28. Markus Lohrey, Word problems on compressed words ICALP 2004
29. Makanin, G.S., The problem of solvability of equations in a free semigroup, *Mat. Sb.*, Vol. 103,(145), 147-233, 1977. English transl. in *Math. U.S.S.R.* Sb. Vol 32, 1977.
30. U. Manber, A text compression scheme that allows fast searching directly in the compressed file, *ACM Transactions on Information Systems*, 15(2), pp. 124-136, 1997
31. N. Markey, Ph. Schnoebelen, A P-Time complete matching problem for SLP-compressed words, IPL 2004
32. M. Miyazaki, A. Shinohara, M. Takeda, An improved pattern-matching algorithm for strings in terms of straight-line programs, Journal of Discrete Algorithms, Vol.1, pp. 187-204, 2000.
33. W. Plandowski, Satisfiability of word equations with constants is in P-Space, JACM 2004
34. W. Plandowski, Testing equivalence of morphisms on context-free languages, *Proceedings of the* 2^{nd} *Annual European Symposium on Algorithms* (ESA'94), LNCS 855, Springer-Verlag (1994), pp. 460–470.
35. W. Plandowski, W. Rytter, Complexity of compressed recognition of formal languages, in *"Jewels forever"*, Springer Verlag 1999 (ed. J. Karhumaki)
36. W.Plandowski, W.Rytter, Applying Lempel-Ziv encodings to the solution of word equations, *ICALP* 1998
37. W. Rytter, Application of Lempel-Ziv Factorization to the Approximation of Grammar-Based Compression. TCS 1-3(299): 763-774 (2003), Preliminary version in Combinatorial Pattern Matching, June 2002,
38. W. Rytter, Compressed and fully compressed pattern-matching in one and two-dimensions, Proceedings of IEEE, November 2000, Volume 88, Number 11, pp. 1769-1778