Complexity of Language Recognition Problems for Compressed Words

Wojciech Plandowski * and **Wojciech Rytter** **

Summary. The **compressed recognition** 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 new 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.

Two types of compression are considered: Lempel-Ziv compression and compression in terms of straight-line programs (or sequences of recurrences, or context-free grammars generating single texts). These compressions are polynomially related and most of our results apply to both of them.

Denote by $LZ(w)$ ($SLP(w)$) the version of a string w produced by *Lempel-Ziv encoding* (*straight-line program*). The complexity of the following problem is considered:
given a compressed version ($LZ(w)$ or $SLP(w)$) of the input word w, test the membership $w \in L$, where L is a formal language.
The complexity depends on the type and description of the language L. Surprisingly the proofs of NP-hardness are in this area usually easier than the proofs that a problem in in NP.

In particular the membership problem is in polynomial-time for regular expressions. However it is NP-hard for semi-extended regular expressions and for (linear) context-free languages, and we don't know if it is in NP in these cases. The membership problem is NP-complete for unary regular expressions with compressed constants.

The membership problem is in $DSPACE(n^2)$ for general context-free sets L and in $NSPACE(n)$ for linear languages. We show that for unary languages compressed recognition of context-free languages is NP-complete.

We also briefly discuss some known results related to the membership problem for the string-matching languages and for languages related to string-matching: square-free words, squares, palindromes, and primitive words.

* Instytut Informatyki, Uniwersytet Warszawski, Banacha 2, 02–097 Warszawa, Poland. Email:wojtekpl@mimuw.edu.pl.

** Instytut Informatyki, Uniwersytet Warszawski, Banacha 2, 02–097 Warszawa, Poland, and Department of Computer Science, University of Liverpool. Supported partially by the grant KBN 8T11C03915. Email:rytter@mimuw.edu.pl

1 Introduction

One of the basic language recognition problems is the *string-matching* problem:

check if $P\#T \in L_{string-match}$,

where

$L_{string-match} = \{x\#y \; : \; x$ is a subword of y, and $x,\ y$ do not contain $\#\}$.

The *compressed string-matching* problem (when the input is given in the compressed form) has been investigated in [1], [2], [3], [12], [13], [10]. The *fully compressed* matching occurs when both P and T are given in compressed forms.

However other typical membership problems have not been considered, e.g. recognition of context-free languages.

In this paper we discuss the complexity of several language recognition problems for input words given in a compressed form. We also briefly discuss the complexity of the compressed string-matching.

1.1 Lempel-Ziv encodings

The LZ compression (see [15]) gives a very natural way of representing a string and it is a practically successful method of text compression. We consider the same version of the LZ algorithm as in [11] (this is called $LZ1$ in [11]). Intuitively, LZ algorithm compresses the text because it is able to discover some repeated subwords. We consider here the version of LZ algorithm without *self-referencing* but our algorithms can be extended to the general self-referential case. Assume that Σ is an underlying alphabet and let w be a string over Σ. The factorization of w is given by a decomposition:

$$w = c_1 f_1 c_2 \dots f_k c_{k+1},$$

where $c_1 = w[1]$ and for each $1 \le i \le k$ $c_i \in \Sigma$ and f_i is the longest prefix of $f_i c_{i+1} \dots f_k c_{k+1}$ which appears in $c_1 f_1 c_2 \dots f_{i-1} c_i$.

We can identify each f_i with an interval $[p,q]$, such that $f_i = w[p..q]$ and $q \le |c_1 f_1 c_2 \dots f_{i-1} c_{i-1}|$. If we drop the assumption related to the last inequality then it occurs a *self-referencing* (f_i is the longest prefix which appears before but not necessarily terminates at a current position). We assume that this is not the case.

Example.
The factorization of a word *aababbabbaababbabba#* is given by:

$$c_1\ f_1\ c_2\ f_2\ c_3\ f_3\ c_4\ f_4\ c_5 = a\ a\ b\ ab\ b\ abb\ a\ ababbabba\ \#.$$

After identifying each subword f_i with its corresponding interval we obtain the LZ encoding of the string. Hence

$$LZ(aababbabbababbabb\#) = a[1,1]b[1,2]b[4,6]a[2,10]\#.$$

1.2 Straight-line programs encodings

In [17] and [10] the compressed strings were considered in terms of straight-line programs (context-free grammars generating single texts). A *straight-line program* (SLP for short) $\mathcal{R}$ is a sequence of assignment statements:

$$X_1 = expr_1;\ X_2 = expr_2; \ldots;\ X_n = expr_n,$$

where X_i are variables and $expr_i$ are expressions of the form:

- $expr_i$ is a symbol of a given alphabet Σ, or
- $expr_i = X_j \cdot X_k$, for some $j, k < i$, where $\cdot$ denotes the concatenation of X_j and X_k.

Example.
The following program $\mathcal{T}$ describes the $8th$ *Fibonacci word*, see [16].

$$X_1 = \mathtt{b};\ X_2 = \mathtt{a};\ X_3 = X_2 \cdot X_1;\ X_4 = X_3 \cdot X_2;$$
$$X_5 = X_4 \cdot X_3;\ X_6 = X_5 \cdot X_4;\ X_7 = X_6 \cdot X_5;\ X_8 = X_7 \cdot X_6,$$

Both types of encodings are polynomially related, so from the point of view of polynomial-time computability we can use term *compressed* to mean LZ or SLP compression.

Theorem 1.
(a). *Let $n = |LZ(w)|$. Then we can construct a context-free grammar G of size $O(n^2 \log n)$ which generates w and which is in the Chomsky normal form. Moreover the height of the derivation tree of w with respect to G is $O(n \cdot \log\log n)$.*
(b). *Assume w is generated by a straight-line program of size n then $|LZ(w)| = O(n)$.*

2 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 will be a part of the input size.

Theorem 2.
(a) *We can decide for SLP-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 SLP-compressed words the membership in a language described by given determiniatic automaton M in $O(n \cdot m)$ time, where m is the number of states of M.*

Proof.
(a) Construct a nondeterministic finite automaton M accepting the language described by W with $O(m)$ states. For each variable X denote by $TransTable_X$ a relation on states of M such that (p, q) are in this relation iff there is a path from the state p to q when reading the input word generated by X.

If we have an equation $X = Y \cdot Z$ then

$$TransTable_X = TransTable_Y \otimes TransTable_Z,$$

where $\otimes$ is the boolean matrix multiplication.

In this way we compute bottom-up the transition table for X_n. The most costly operation is the multiplicaton $\otimes$ of boolean matrices, which can be done in $O(m^3)$ time.

(b) In this case the table $TransTable_X$ becomes a function, which can be represented by a vector. Composition of such functions can be done in $O(m)$ time, so the total cost is $O(n \cdot m)$. This completes the proof.

We use the following problem 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 1. *The problem SUBSET SUM is NP-complete.*

Proof. see [9], and [8], pp. 223.

Theorem 3.
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.

It is not obvious if the previously considered problem is in NP for regular expressions containg the operation *, in this case there is no polynomial bound on the length of accepting paths of A. There is a simple argument in case of unary languages.

In the next theorem we show an interesting application of the Euler path technique to a unary language recognition.

Theorem 4. *[unary expressions]*
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.

Proof.
Let A be a nondeterministic finite automaton accepting a given regular expression. We can assume that there is only one accepting state, the initial state has zero indegree and the accepting state has zero outdegree. Let S be a set of edges of A. Denote by $in(S, q)$ and $out(S, q)$ the number of edges in S entering (outgoing) a given state a of A. Denote by q_0, q_a the initial (accepting) state of A.

We say that a multiset S is *valid* for A iff it is a set of edges of an accepting path of A.

Claim. (Euler Path Property)
S is valid for A iff the following conditions are satisfied:
(a) $in(S, q) = out(S, q)$ for each noninitial and nonacepting state of A,
(b) $in(S, q_0) = out(S, q_a) = 0$ and $out(S, q_0) = in(S, q_a) = 1$.
(c) the graph which is formed from the edges in S is connected,

Proof. (of the claim)
Obviously each accepting path corresponds to a valid multiset S. On the other hand if S is valid and if we continue arbitrarily a path starting at q_0 and using each time a single edge from S (deleting it after usage) then the only possibility is to finish in q_a. In this way an accepting path corresponding to S is constructed in a similar way as an Euler path in a graph in the proof of Euler theorem.

We can guess a multiset S since number of edges of A is linear. Knowing the multiplicity of each edge we can now deterministically check if the total length of all edges in the multiset S (counting with multiplicity) equals the length of the input word. We have numbers with polynomial number of bits, hence all computations are in polynomial time.

Theorem 5.
The problem of checking membership of a compressed word in a language described by a semi-extended regular expression is NP-hard.

Proof.
We use the following number-theoretic fact.

`Chinese Remainder Theorem`.

Let $p_1, p_2, \ldots, p_k$ be relatively prime integers. Then for each sequence $r_1, r_2, \ldots, r_k$ there is exactly one integer $0 \leq a < p_1 p_2 \ldots p_k$ such that $a \; mod \; p_i \; = \; r_i$ for each $1 \leq i \leq k$.

We use Thorem 3. Assume the total number of bits representing compressed unary constants and size of the input in this theorem is m. Then we replace the compressed strings d^{a_i} by $(d^{p_1})^* d^{r_1} \cap (d^{p_2})^* d^{r_2} \cap ...(d^{p_k})^* d^{r_k}$, where p_i are all prime numbers smaller than m and $r_j = a_i \; mod \; p_j$.

3 Compressed recognition problems for context-free languages

Theorem 6.
The problem of checking membership of a compressed word in a given linear cfl L is NP-hard, even if L is given by a context-free grammar of a constant size.

Proof.
Take an instance of the subset-sum problem with the set $A = \{a_1, a_2, \ldots, a_n\}$ of integers and an integer K. Define the following language:

$$L = \{d^R \$ d^{v_1} \# d^{v_2} \# \ldots \# d^{v_t} \; : \; t \geq 1 \text{ and there is a subset } A' \subseteq \{v_1, \ldots, v_t\} \text{ such that } \textstyle\sum_{u \in A'} u \; = \; R\} \; ?$$

L is obviously a linear context-free language generated by a linear context-free grammar of a constant size. We can reduce an instance of the subset sum-problem to the memebership problem:

$$d^K \$ d^{a_1} \# d^{a_2} \# \ldots \# d^{a_n} \; \in \; L.$$

Theorem 7.
(a) *The problem of checking membership of a compressed word in a given linear cfl is in $NSPACE(n)$.*
(b) *The problem of checking membership of a compressed word in a given cfl is in $DSPACE(n^2)$.*

Proof.
We can easily compute in linear space a symbol on a given position in a compressed input word. Now we can use a space-efficient algorithm for the recognition of context-free languages. It is known that linear languages can be recognized in $O(logN)$ nondeterministic space and general cfls can be done in $O(log^2N)$ deterministic space, where N is the size of the uncompressed input word. In our case $N = O(2^n)$, this gives required $NSPACE(n)$ and $DSPACE(n^2)$ complexities.

Theorem 8.
The problem of checking membership of a compressed unary word in a given cfl is NP-complete.

Proof.
(1) The proof of NP-hardness.
We use the construction from Theorem 3 and construct a grammar generating a language described by a *star-free* regular expression. Each compressed unary constant can be generated by a polynomial size grammar since there is grammar generating d^{2^k} with $O(k)$ productions.

(2) Containment in NP.
Assume the grammar G is in Chomsky normal form. Let M be a multiset of productions of the grammar G. We say that M is *valid* for G iff there is a derivation tree T of G such that M is the multiset of productions used in T.

Denote by $\#fathers_A(M)$ the number of occurrences of a nonterminal A as a root in a production in M, and by $\#sons_A(M)$ the number of occurrences of a nonterminal A as a son in a production in M. Let S be the starting nonterminal of G.

Claim.
A multiset M of productions is valid for G iff the following conditions are satisfied:
(a) $\#sons_A(M) \ = \ \#fathers_A(M)$ for each nonterminal $A \neq S$;
(b) $\#sons_S(M) \ = \ \#fathers_S(M) - 1$.
(c) each nonterminal occuring in productions in M is reachable from S using the productions in M.

Proof. (of the claim)
If T is a derivation tree then the multiset of productions in T certainly satisfies the conditions. Conversely, assume M satisfies both conditions. Then we can construct the tree top-down starting with a production which has on the left side S and belongs to M. Whenever we have a leaf of a partially constructed tree which is a nonterminal we can expand it using a production from M. Each time we use a production from M we delete it from M. The conditions guarantee that we finish successfully.

We can guess a multiset M, check its validity and check if the number (counting multiplicites) of terminal productions (of the form $A \rightarrow d$) equals the length of the input unary word. This completes the proof.

Observe here that we cannot strengthen Theorem 8 as in Theorem 6 by bounding the size of the input grammar by a constant since for constant-size grammars and unary alphabets we can transform the grammar to a finite automaton of a constant size and obtain a deterministic polynomial time algorithm for the problem, due to Theorem 2.

4 Languages related to string-matching

The key role in string-matching problems is the size of the description of an exponential size set of positions. Small descriptions use the fact that exponential length arithmetic progression can be identified with three integers. A set of integers forming an arithmetic progression is called here *linear*. We say that a set of positive integers from $[1 \ldots U]$ is *linearly-succinct* iff it can be decomposed in at most $\lfloor \log_2(U) \rfloor + 1$ linear sets. Denote $Periods(w) = \{p : p \text{ is a period of } w\}$. The following lemma was shown in [10].

Lemma 2 (linearly-succinct sets lemma).
The set $Periods(w)$ is linearly-succinct.The set of occurrences of a word overlapping a fixed position is linear.

Surprisingly the set of occuurences of a fixed pattern in a compressed text does not necessarily have a linearly succinct representation, which shows nontriviality of the compressed string-matching.

Example 1. The set of all occurrences of a given pattern in a "well" compressible text T is not necessarily linearly-succinct, even if the pattern is a single letter. Consider the recurrently defined sequence of words $\{a_i\}_{i \geq 0}$ which is defined in the following way:

$$a_0 = a \quad a_i = a_{i-1} b^{|a_{i-1}|} a_{i-1} \text{ for } i \geq 1.$$

Let S_i be the set of positions of occurrences of the letter a in the word a_i. Clearly, $|S_i| = 2^i$. It is not difficult to prove that each arithmetic sequence in S_i has length at most 2. This means that each decomposition of the set S_i into arithmetic sequences contains at least $|S_i|/2 = 2^{i-1}$ sequences and the set S_i is not linearly-succinct. Note, that the words a_i are "well" compressible since $|LZ(a_i)| \leq 4i + 1$ and $|a_i| = 3^i$.

The definition of words a_i resembles the definition of Cantor's set which is the set of points in interval $[0, 1]$ remaining after applying the following procedure. Divide all existing intervals into three equal parts and remove the middle one. In the definition of a_i we divide a_i into three equal parts and the middle part is filled by letters b. We do the same recurently with remaining parts of a_i. What remains after applying this procedure is replaced by a i.e. the set of positions of a corresponds to Cantor's set. Indeed, those two sets satisfy similar dependence. Cantor's set consists of points ternary expansion of which consists of zeroes and twos. Similarly positions of letters a in a_i consists of zeroes and twos in ternary expansions under the assumption that the position of the first letter is zero not one.

In the example occurrences of a pattern are well compressible by the word a_i itself if we assume that the symbol a represents a starting position of an occurrence of the pattern a in a_i and the symbol b represents a position which is not a starting position of an occurrence of the pattern a in a_i. It appears that this is always the case. Let p be a pattern and t be a compressed text. Let $occ(p,t)$ be a word of length $|t|$ over the alphabet $\{0,1\}$ such that $occ[i]=1$ iff i is an ending position of an occurrence of p in t.

Theorem 9. *There is SLP for $occ(p,t)$ which is of polynomial size with respect to SLP for t and to $|p|$.*

Proof. First we construct a deterministic automaton $\mathcal{A}$ accepting words with suffix p [7]. The automaton has $|p|$ states. Let $\mathcal{P}$ be SLP for t. We construct a SLP $\mathcal{R}$ for the path in $\mathcal{A}$ labeled t which starts in the initial state q_0 of $\mathcal{A}$. The constants in $\mathcal{R}$ are edges of $\mathcal{A}$. The variables in $\mathcal{R}$ are triples (q,X,q') where X is a variable from SLP for t and q' is the state reached from q after reading the word corresponding to X. The variable (q,X,q') generates the path in $\mathcal{A}$ labeled the word corresponding to X which goes from q to q'. The construction of $\mathcal{R}$ is as follows:

- $(q,X,q')=(q,a,q')$ iff $X=a$ in SLP for t and (q,a,q') is an edge in $\mathcal{A}$,
- $(q,V,q')=(q,X,q'')(q'',Y,q')$ iff $V=XY$ in SLP for t and q' is reachable by S starting from q and q'' is reachable by X starting from q.

The variable of the form (q_0,S,q') generates the path for t if S generates t in SLP for t. To obtain SLP for $occ(p,t)$ we replace each edge (q,a,q') in $\mathcal{R}$ by 1 if q' is accepting state in $\mathcal{A}$ and by 0 otherwise.

Let U denote the size of uncompressed text.

Theorem 10. *[13]*
The Fully Compressed Matching Problem can be solved in $O((n\log n)^5\log^4 U)$ time.
We can compute within the same complexity the period of the compressed text and check if it is primitive.

Algorithms for compressed palindromes and squares use ideas from [4]: palindromes are searched using **periodicities** implied by sequences of many palindromes which are *close to each other* and searching of squares is reduced to multiple application of pattern-matching.

Theorem 11. *[13]*
We can check if a text is square-free in $O((n \log n)^6 \log^5 U)$ *time.*

Theorem 12. *[13]*
The compressed representation of all palindromes in the compressed text can be computed in $O((n \log n)^5 \log^4 U)$ *time.*

5 Open problems

There are several interesting questions remaining:

1. Is the problem of compressed recognition for semi-extended regular expressions in NP ? We conjecture it is.
2. Is the problem of compressed recognition for regular expressions with compressed constants in NP ? We conjecture it is.
3. Is the problem of compressed recognition for regular expressions with compressed nonperiodic constants in P ? We conjecture it is.
4. Is the problem of compressed recognition for context-free languages in NP ? We conjecture that this problem is *P-SPACE*-hard.
5. Does the set of all occurrences of the letter a in the n-th Fibonacci word consist of a polynomial number of arithmetic progressions ?
6. Does there exist SLP for $occ(p, t)$ which is of polynomial size with respect to a SLP for t and SLP for p?

References

1. A. Amir and G. Benson, Efficient two dimensional compressed matching, *Proc. of 2nd IEEE Data Compression Conference*, pp. 279–288, March 1992.
2. A. Amir, G. Benson, and M. Farach, Let sleeping files lie: Pattern matching in Z-compressed files, *Proc. of 5th Annual ACM-SIAM Symposium on Discrete Algorithms*, January 1994.
3. A. Amir, G. Benson, and M. Farach, Optimal two-dimensional compressed matching, *Proc. of 21st International Colloquium on Automata, Languages, and Programming*, July 1994.
4. A. Apostolico, D. Breslauer, Z. Galil, Optimal parallel algorithms for periods, palindromes and squares, in *ICALP'92*, 296-307
5. P. Berman, M. Karpiński, L. Larmore, W. Plandowski, and W. Rytter, The complexity of pattern matching of highly compressed two-dimensional texts, Proc. of *8th Annual Symposium on Combinatorial Pattern Matching*, CPM'97, July 1997.
6. B.S. Chlebus and L. Gąsieniec, Optimal pattern matching on meshes, *Proc. 11th Symposium on Theoretical Aspects of Computer Science*, 1994, pp. 213–224.
7. M. Crochemore, W. Rytter, *Text algorithms*, Oxford University Press, 1994.
8. M.R. Garey and D.S. Johnson, Computers and Intractability: A Guide to the Theory of NP-Completeness", W.H. Freeman, 1979

9. R.M. Karp, Reducibility among combinatorial problems, in "Complexity of Computer Computations", Plemum Press, New York", 1972 (editors R.E. Miller and J.W. Thatcher)
10. M. Karpiński, W. Rytter, and A. Shinohara, A pattern matching for strings with short description, *Proc. 6th Combinatorial Pattern Matching*, June 1995.
11. M. Farach and M. Thorup, String Matching in Lempel-Ziv Compressed Strings, *Proc. 27th ACM Symposium on Theory of Computing*, pp. 703–713, 1994.
12. L. Gąsieniec, M. Karpiński, W. Plandowski, and W. Rytter, Randomised efficient algorithms for compressed strings: the finger-print approach, *Proc. 7th Combinatorial Pattern Matching*, pp. 39–49, LNCS 1075, June 1996.
13. L. Gąsieniec, M. Karpiński, W. Plandowski, and W. Rytter, Efficient algorithms for Lempel-Ziv encoding, *Proc. 5th Scandinavian Workshop on Algorithms Theory*, pp. 392–403, LNCS 1097, July 1996.
14. L. Gąsieniec, W. Rytter, Fully compressed pattern matching for LZW encoding, to appear at IEEE Data Compression Conference (1999)
15. A. Lempel and J. Ziv On the complexity of finite sequences, *IEEE Transactions on Information Theory*, pp. 22:75–81, 1976.
16. M. Lothaire, *Combinatorics on Words*, Addison-Wesley, 1983.
17. M. Miyazaki , A. Shinohara, and M. Takeda, An Improved Pattern Matching for Strings in Terms of Straight-Line Programs, *8th Combinatorial Pattern Matching*, July 1997, LNCS 1264, pp. 1–11.
18. W. Plandowski and W. Rytter, Application of Lempel-Ziv encodings to the solution of words equations, *Proc. of 25th International Colloquium on Automata, Languages, and Programming*, to appear.
19. T.A. Welch, A technique for high performance data compression, *IEEE transactions on Computers*, pp. 17:8–19, 1984.
20. J. Ziv and A. Lempel, A universal algorithm for sequential data compression, *IEEE Transactions on Information Theory*, pp. IT-23(3):337–343, 1977.