On the Computational Complexity of Embedding of Compressed Texts

Diploma Thesis of Yury Lifshits

June 2005

Abstract

In this work we consider a well-known problem of processing of compressed texts. We study the following question (called **Embedding**): whether one compressed text is a subsequence of another compressed text? In this paper we show that **Embedding** is $\mathcal{NP}$- and co-$\mathcal{NP}$-hard.

1 Introduction

The size of information flows is quickly growing. Therefore processing of different compressed objects becomes important. Compressed trees, pictures, charts are studied now. One of the central notions is a text compression. In 1977 Lempel and Ziv proposed their model of compression [**?**] which is now classical. In ninetieth in many works a grammar-based compression (straight-line programs) was studied. Here we also investigate the straight-line programs.

1.1 Main notions and problems

We consider a finite alphabet Σ. A word is any finite sequence of its letters. The set of all words is traditionally denoted by Σ^*.

The basic notion of our work is *straight-line program*. Roughly, straight-line program is a context-free grammar generating only one word.

Definition 1. A straight-line program (SLP) is a context-free grammar $\mathcal{P}$ with ordered non-terminal symbols $X_1, \ldots, X_m$ (X_m is a starting symbol)

such that there is only one production for each symbol: either $X_i \to a$, where a is a terminal, or $X_i \to X_j X_k$ for some $j, k < i$.

For straight-line program $\mathcal{P}$ we denote by $eval(\mathcal{P})$ the unique word generated by $\mathcal{P}$. Therefore $eval(\mathcal{P}) = eval(X_m)$.

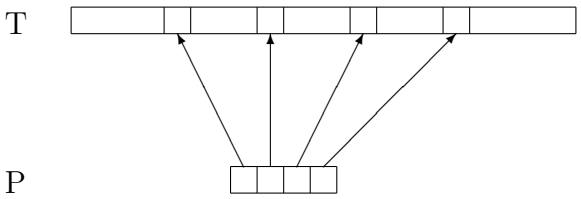
Let us explain the origin of the term "straight-line program". The reason is that any text presented by a grammar ofthe type described above could be generated by a program using only one operator — assignment statement.

The main problems about compressed texts are the following:

- **Compressed Equivalence**: given two compressed texts, determine whether they are equal or not.
- **Compressed Pattern Matching**: given a pattern and a compressed text, determine whether the pattern is a subword of the text.
- **Fully Compressed Pattern Matching**: given a **compressed** pattern and a compressed text, determine whether the pattern is a subword of the text.
- **Membership**: let a language L be fixed. Given a compressed word, determine whether it belongs to L.

Many papers deal with these problems [**?**, **?**, **?**, **?**]. First three problems have polynomial complexity. In the case of regular language, **Membership** is also solvable in polynomial time. But **Membership** for context-free language is already $\mathcal{PSPACE}$-complete [**?**]. After studying these basic questions, one started to search for effective algorithms for other problems. The following problem is very close to **Fully Compressed Pattern Matching**:

- **Compressed Subsequence Matching** (for short — **Embedding**). Given a compressed string P (pattern) and a compressed string T (text), determine whether letters of P form a subsequence of T (we denote this property by $P \hookrightarrow T$).



Till now there was no complexity bounds for **Embedding**. It might be solvable in polynomial time as well as $\mathcal{PSPACE}$-complete. In present work we obtain lower bounds showing that this problem is $\mathcal{NP}$-hard and even (assuming that $\mathcal{NP} \neq co-\mathcal{NP}$) does not belong to $\mathcal{NP}$.

1.2 Related work

In the paper [**?**], based on [**?**], a polynomial algorithm for **Fully Compressed Pattern Matching** was constructed. Also, authors presented a polynomial algorithm for **Membership** in the case of regular language.

It was showed recently [**?**] that **Membership** is $\mathcal{P}$-complete in the case of regular language.

Another important result was obtained in [**?**]. For any given text an approximately shortest SLP generating this text was constructed. More formally, there was constructed an algorithm working in time $O(n \cdot \log|\Sigma|)$ which for any given text of length n computes $O(\log n)$-approximation of minimal SLP generating this text. This means that the adequate (close to the minimal possible) compression in a grammar-based model could be done efficiently.

Next, we should mention [**?**], where **Compressed Pattern Matching** problems were extended to the two-dimensional case. It was found that complexity increases by such generalization. **Compressed Pattern Matching** is $\mathcal{NP}$-complete while **Fully Compressed Pattern Matching** is Σ_2^P-complete.

Another generalization was considered in [**?**]. Recall that strings are elements of a free finitely generated monoid. In this paper processing of compressed monoid elements for various classes of finitely generated monoids were studied. Depending on the nature of monoid, the completeness results for **Compressed Equivalence** were obtained for complexity classes $\mathcal{P}$, co-$\mathcal{NP}$, $\mathcal{PSPACE}$ and $\mathcal{EXPSPACE}$.

Recently, applications for algorithms on compressed texts in analysis of message sequence charts were found, see [**?**].

We refer to [**?**, **?**] for the surveys on compressed texts processing.

1.3 Motivation and our results

In the case of normal (uncompressed) texts the complexity of subsequence matching is linear, hence this problem is not harder than pattern match-

ing. After discovering of the polynomial algorithm for pattern matching in compressed texts, The problem of existence of analogous algorithm for subsequence matching becomes quite natural. Using the standard assumption $\mathcal{P} \neq \mathcal{NP}$, we prove in our paper that such algorithm does not exist.

1.4 Scheme of proofs

There are two main results in the paper. We start with a (many-one) reduction of the known $\mathcal{NP}$-complete problem **Subset Sum** to **Embedding**. Hence we obtain $\mathcal{NP}$-hardness of the latter. The next step is a reduction of **Embedding** to **Non-embedding** and vice versa. Co-$\mathcal{NP}$-hardness of our problem is immediate corollary of such symmetry.

2 $\mathcal{NP}$-hardness

Let us recall the well-known $\mathcal{NP}$-complete problem **Subset Sum** (see [**?**]):

- Given integers $w_1, \ldots, w_n, t$ in binary form, determine whether there exist $x_1, \ldots, x_n \in \{0, 1\}$ such that $\sum_{i=1}^{n} x_i \cdot w_i \quad = \quad t$.

Theorem 1. *There is a reduction (in the sense of Karp) from* **Subset Sum** *to* **Embedding**.

Proof. Let $t, \bar{w} = \langle w_1, \ldots, w_n \rangle$ be input data for **Subset Sum** (let us consider only $n > 1$). We are going to construct straight-line programs $\mathcal{F}$ and $\mathcal{G}$ such that there exists a subset of $\bar{w}$ with sum equal to t iff $eval(\mathcal{F}) \hookrightarrow eval(\mathcal{G})$.

We begin with some notation. Let $s = w_1 + \cdots + w_n$ and $N = 2^n s$. For every subset of $\bar{w}$ there is a corresponding string $x = \overline{x_1 \ldots x_n}$ of length n consisting of zeroes and ones and therefore also a corresponding integer in the range from 0 to $2^n - 1$. Let us denote $x \circ \bar{w} = \sum_{i=1}^{n} x_i w_i$, in fact, $x \circ \bar{w}$ is a sum of the subset corresponding to x.

Step 1: construction of texts G and F.

$$G = G_0^{5N} \qquad\qquad G_0 = G_1 G_2 G_3 G_4$$

$$G_1 = \prod_{x=0}^{2^n - 1} (10^s) = (10^s)^{2^n} \qquad\qquad G_2 = 0^{2N}$$

$$G_3 = \prod_{x=0}^{2^n-1} (0^{x\circ\bar{w}} 10^{s-x\circ\bar{w}}) \qquad\qquad G_4 = 0^{t+1}$$

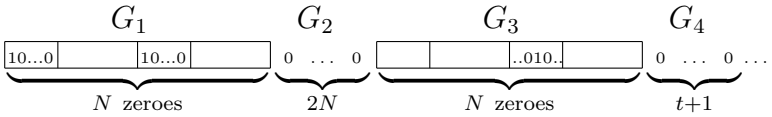
$$F = F_0^{5N-1} \qquad\qquad F_0 = 10^{3N+t}10^{N+1}$$

We use symbol $\prod$ to denote the concatenation of corresponding words performed in order of index changing from the lower bound to the upper one.

By these equalities we define texts F and G. Below we will prove that they could be generated by straight-line programs $\mathcal{F}$ and $\mathcal{G}$. But before let us prove that embedding of F into G is equivalent to the existence of a subset of $\bar{w}$ with sum equal to t.

Step 2: equivalence of embedding and existence of subset with required sum.

"Subset exists $\Rightarrow$ embedding holds". Let x be an integer corresponding to a subset of $\bar{w}$ with sum equal to t, i.e., $x \circ \bar{w} = t$. Consider the beginning of G, that is $G_1G_2G_3G_4G_1\ldots$ and take the following embedding of F_0 in G: take 1 in x-th block of G_1, then take $3N + t$ zeroes, then (exactly because $x \circ \bar{w} = t$) the next letter is 1. Taking following $N + 1$ zeroes, we find ourselves before x-th one in the second copy of G_1. Thus having $5N$ groups $G_0 = G_1G_2G_3G_4$ we can embed $5N - 1$ copies of F_0. And that is what we need to prove: $F \hookrightarrow G$.

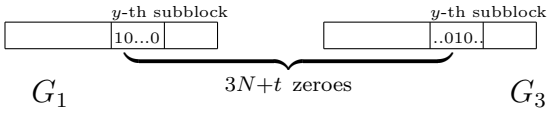


"Embedding holds $\Rightarrow$ subset exists". We have $F \hookrightarrow G$. Our aim is to prove that the answer to **Subset Sum** problem is positive. Assume the converse. Then we will find a contradiction by counting zeroes in F and G, as follows.

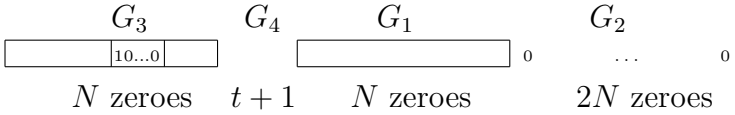
Not every zero in G is an embedding image of some zero in F. Let us estimate the total number of such unused zeroes. Our embedding consists of $5N - 1$ disjoint embeddings of F_0 in G.

There are two ones in F_0 and there are exactly $3N + t$ zeroes between them. We claim that there is no such pair of ones (with exactly $3N + t$ ones between them) in G. Assume the converse, that there exists such a pair. Consider two cases depending on the location of the left one.

If this one is in G_1 block (say subblock number y), then after shifting by $3N+t$ zeroes to the right we come to the y-th subblock in G_3. If after t zeroes in this subblock there is a one, then $y \circ \bar{w} = t$. Hence we get a contradiction with assumption of negative answer to **Subset Sum** problem.



In the second case the left one in our pair is situated in G_3. Then shifting by $3N + t$ zeroes to the right we get into the block G_2 where is no ones at all.



Therefore in each embedding of F_0 in G between two ones there is at least one zero which is not used. To complete the proof we estimate the number of zeroes in G from two sides. By construction, this number is equal to $5N \cdot (4N+t+1)$. From the other hand, each F_0 contains $4N+t+1$ zeroes and also there are at least $5N-1$ unused zeroes in G. Putting all together, total number of zeroes in G must be greater than or equal to

$$(5N-1)\cdot(4N+t+1)+5N-1 = 5N\cdot(4N+t+1)+(N-t-2) > 5N\cdot(4N+t+1).$$

Let us explain the last inequality: $N = s2^n \geq 4s > t+2$. Thus the number of zeroes in G must be greater than the actual quantity. This contradiction completes the proof.

Step 3: generation of F and G by straight-line programs. Note that with one exception F and G are constructed from 0 and 1 only by a polynomial number of concatenations and raising to a power (in our construction all power degrees have polynomial length in binary form). These constructions could be directly realized by SLP. The only nontrivial block is G_3. A generation of G_3 by SLP of polynomial size was proposed for first time by M. Lorhey [**?**]. We repeat it here for completeness.

The SLP generating

$$G_3 = \prod_{\bar{x}=0}^{2^n-1} (0^{x\circ\bar{w}}10^{s-x\circ\bar{w}}).$$

is as follows:

$$S_1 \to 10^{s+w_1}1$$
$$S_{k+1} \to S_k 0^{s-s_k+w_{k+1}} S_k.$$

Let S_n be a starting symbol. We prove that $eval(S_n) = G_3$.

Claim: $eval(S_k) = \left(\prod_{\bar{x}\in\{0,1\}^k \setminus \{\bar{1}_k\}} (0^{\bar{x}\cdot\bar{w}_k} 10^{s-\bar{x}\cdot\bar{w}_k}) \right) 0^{s_k}1.$

We now prove it by induction. For $k = 1$ our claim means that $10^{s+w_1}1 = 0^0 10^{s-0}0^{s_1}1$ which is true. For $k+1 \leq n$ we get the following chain of equalities:

$$eval(S_{k+1}) = \left(\prod_{\bar{x}\in\{0,1\}^{k+1} \setminus \{\bar{1}_{k+1}\}} (0^{\bar{x}\cdot\bar{w}_{k+1}} 10^{s-\bar{x}\cdot\bar{w}_{k+1}}) \right) 0^{s_{k+1}}1 =$$

$$\underbrace{\left(\prod_{\bar{x}\in\{0,1\}^k} (0^{\bar{x}\cdot\bar{w}_k} 10^{s-\bar{x}\cdot\bar{w}_k}) \right)}_{eval(S_k)0^{s-s_k}} \underbrace{\left(\prod_{\bar{x}\in\{0,1\}^k \setminus \{\bar{1}_k\}} (0^{\bar{x}\cdot\bar{w}_k + w_{k+1}} 10^{s-\bar{x}\cdot\bar{w}_k - w_{k+1}}) \right) 0^{w_{k+1}}}_{0^{w_{k+1}} eval(S_k)} 0^{s_k}1 =$$

$$eval(S_k)0^{s-s_k+w_{k+1}} eval(S_k) = eval(S_{k+1}),$$

which proves the claim.

To complete the proof take our claim for $k = n$:

$$eval(S_n) = \prod_{\bar{x}\in\{0,1\}^n} \left(0^{\bar{x}\cdot\bar{w}} 10^{s-\bar{x}\cdot\bar{w}}\right) = G_3.$$

Step 4: reduction could be done in polynomial time. The last thing to prove is that the construction of SLP generating F and G from the input data of **Subset Sum** problem could be done in polynomial time. Note that in construction of SLP generating F and G we use only concatenations and rising to a power. All power degrees are results of arithmetical operations on t and $w_1, \ldots, w_n$. □

Corollary 1. Embedding *is* $\mathcal{NP}$*-hard.*

3 co-$\mathcal{NP}$-hardness

Theorem 2. *There are reductions (in the sence of Karp) of* **Embedding** *to* **Non-embedding** *and of* **Non-embedding** *to* **Embedding**.

Proof. We now prove the following: there exists a polynomial algorithm that for any given compressed texts F and G produces compressed texts F_1 and G_1 such that

$$F \hookrightarrow G \Leftrightarrow F_1 \not\hookrightarrow G_1. \qquad (*)$$

Naturally that their total (compressed) size is at most polynomially larger than that of F and G.

In the case of unary alphabet both problems belong to $\mathcal{P}$. Therefore we consider only the alphabets containing at least two letters. Note that we can compute last letter of F in polynomial time and add to the end of G a different letter (getting G'). Still after this $F \hookrightarrow G \Leftrightarrow F \hookrightarrow G'$. Thus, without loss of generality, we can consider only the cases where the last letters of F and G are different.

Let $F = f_1 \dots f_k$, $G = g_1 \dots g_m$. For every letter of alphabet a we denote $X_a = (\Sigma/a)^{m+1}$, where (Σ/a) is a concatenation of all letters of alphabet except a in some (arbitrary) order.

Construction:

$$F_1 = G = g_1 \dots g_m$$

$$G_1 = X_{f_1} f_1 \dots X_{f_k} f_k$$

Proof of property $(*)$: represent G in the following form (clearly it is unique): $G = R_1 f_1 \dots R_l f_l R_{l+1}$, where R_i does not contain letter f_i. The statement $F \hookrightarrow G$ is equivalent to the equality $l = k$ in our representation.

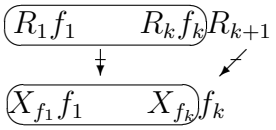
If $l < k$, then $F_1 \hookrightarrow G_1$. Actually for every $1 \le i \le l+1$ it is true that $R_i \hookrightarrow X_{f_i}$ and $f_i = f_i$.

$$\begin{array}{ll} R_1 f_1 R_2 X_2 & R_l f_l R_{l+1} \\ \downarrow \ \downarrow & \downarrow \downarrow \downarrow \\ X_{f_1} f_1 & X_{f_l} f_l X_{f_{l+1}} \dots \end{array}$$

Let now $l = k$. By assumption g_m is not equal to f_k. Then $R_{l+1} \neq \varnothing$. We claim that

$$R_1 f_1 \dots R_i f_i \not\hookrightarrow X_{f_1} f_1 \dots X_{f_i}. \qquad (**)$$

Our proof goes by induction on i. In the case $i = 1$ this follows from $f_1 \not\hookrightarrow X_{f_1}$. Step $i \to i+1$: assume that there is an embedding for $i+1$. Recall that $f_{k+1} \not\hookrightarrow X_{f_{k+1}}$. Thus f_{k+1} is not embedded to the right from f_k and we also get an embedding $(**)$ for i. Thus we get a contradiction with induction assumption. The claim $(**)$ is proved. Combining $(**)$ for $i = k$ and observation $R_{k+1} \not\hookrightarrow f_k$ we have $F_1 \not\hookrightarrow G_1$.



We now prove that reduction is polynomial. Notice that X_a could be constructed in polynomial (w.r.t. $\log m$) time. To complete the construction of G_1 we add rules like $A \to X_a a$ for every letter in the alphabet and use these non-terminal symbols instead of the corresponding letters in our SLP. □

Corollary 2. Embedding *is co-$\mathcal{NP}$-hard.*

Proof. By Theorem 1 **Embedding** is $\mathcal{NP}$-hard. Since we proved our hardness result using reduction in the sense of Karp, **Non-embedding** is co-$\mathcal{NP}$-hard. Recall that we already constructed a reduction from **Non-embedding** to **Embedding** (Theorem 2). Thus **Embedding** is also co-$\mathcal{NP}$-hard. □

4 Conclusion and open problems

In our paper we considered the problem of subsequence search in compressed texts. We obtained lower bounds for computational complexity of this problem. Specifically, we gave a proof (under assumption that $\mathcal{NP} \neq co - \mathcal{NP}$) that this problem does not belong to $\mathcal{NP}$. From the other side, only trivial upper bound is known: there is an algorithm in $\mathcal{PSPACE}$. The main open problem is to close this gap.

References

[1] Piotr Berman, Marek Karpinski, Lawrence L. Larmore, Wojciech Plandowski and Wojciech Rytter. *On the Complexity of Pattern Matching for Highly Compressed Two-Dimensional Texts*, Journal of Computer and Systems Science, vol. 65, number 2, pp. 332–350, 2002.

[2] M.R. Garey and D.S. Johnson. *Computers and Intractability: a Guide to the Theory of NP-completeness*, Freeman, 1979.

[3] Leszek Gasieniec, Marek Karpinski, Wojciech Plandowski and Wojciech Rytter. *Efficient Algorithms for Lempel-Ziv Encoding*

(Extended Abstract), Proceedings of the 5th Scandinavian Workshop on Algorithm Theory (SWAT 1996), Springer-Verlag, LNCS 1097, pp. 392–403, 1996.

[4] Blaise Genest and Anca Muscholl. *Pattern Matching and Membership for Hierarchical Message Sequence Charts*, In Proceedings of the 5th Latin American Symposium on Theoretical Informatics (LATIN 2002), Springer-Verlag, LNCS 2286, pp. 326–340, 2002.

[5] Markus Lorhey. *Word problems on compressed word*, ICALP 2004, Springer-Verlag, LNCS, 3142, pp. 906–918, 2004.

[6] N. Markey and Ph. Schnoebelen. *A PTIME-complete matching problem for SLP-compressed words*, Information Processing Letters, vol. 90, number 1, pp. 3–6, 2004.

[7] Wojciech Plandowski. *Testing Equivalence of Morphisms on Context-Free Languages*, Second Annual European Symposium on Algorithms (ESA'94), Utrecht (The Netherlands), Springer-Verlag, LNCS 855, pp. 460–470, 1994.

[8] Wojciech Plandowski and Wojciech Rytter. *Complexity of Language Recognition Problems for Compressed Words*, Jewels are Forever, Contributions on Theoretical Computer Science in Honor of Arto Salomaa, Springer-Verlag, pp. 262–272, 1999.

[9] Wojciech Rytter. *Algorithms on Compressed Strings and Arrays*, Proceedings of the 26th Conference on Current Trends in Theory and Practice of Informatics (SOFSEM'99), Springer-Verlag, LNCS 1725, pp. 48–65, 1999.

[10] Wojciech Rytter. *Compressed and fully compressed pattern matching in one and two dimensions*, Proceedings of the IEEE, vol. 88, number 11, pp. 1769–1778, 2000.

[11] Wojciech Rytter. *Application of Lempel-Ziv factorization to the approximation of grammar-based compression*, Theoretical Computer Science, vol. 302, number 1–3, pp. 211–222, 2003.

[12] Wojciech Rytter. *Grammar Compression, LZ-Encodings, and String Algorithms with Implicit Input*, Proceedings of the 31st International

Colloquium on Automata, Languages and Programming(ICALP 2004), Springer-Verlag, LNCS 3142, pp. 15–27, 2004.

[13] J. Ziv and A. Lempel. *A universal algorithm for sequential data compression*, IEEE Transactions on Information Theory, vol. 23, number 3, pp. 337-343, 1977.