Reversing Longest Previous Factor Tables is Hard*

Jing He, Hongyu Liang, and Guang Yang

IInstitute for Interdisciplinary Information Sciences,
Tsinghua University, Beijing, China
{he-j08,lianghy08}@mails.tsinghua.edu.cn, 2006yangguang@gmail.com

Abstract. The Longest Previous Factor (LPF) table of a string s of length n is a table of size n whose i^{th} element indicates the length of the longest substring of s starting from position i that has appeared previously in s. LPF tables facilitate the computing of the Lempel-Ziv factorization of strings [21,22] which plays an important role in text compression. An open question from Clément, Crochemore and Rindone [4] asked whether the following problem (which we call the *reverse LPF problem*) can be solved efficiently: Given a table W, decide whether it is the LPF table of some string, and find such a string if so.

In this paper, we address this open question by proving that the reverse LPF problem is NP-hard. Thus, there is no polynomial time algorithm for solving it unless $P = NP$. Complementing with this general hardness result, we also design a linear-time online algorithm for the reverse LPF problem over input tables whose elements are all 0 or 1.

1 Introduction

The concept of Longest Previous Factor tables is introduced by Crochemore and Ilie [7]. Let s be a string of length n. The Longest Previous Factor (LPF) table of s, denoted by LPF_s, is a table of length n defined as follows: For all $1 \leq i \leq n$, the i-th element of LPF_s (denoted as $LPF_s[i]$) is the length of the longest substring of s beginning at position i that appears previously in s. More formally, we have

$$LPF_s[i] = \max\{k \mid \exists 1 \leq j < i,\ s[i, i+k-1] = s[j, j+k-1]\}, \qquad (1)$$

where $s[i, j]$ denotes the substring of s starting from position i and ending at position j (and it means the empty string if $i > j$).

Using LPF tables one can easily compute the Lempel-Ziv factorization of a string [21,22], which is of great importance in lossless data compression and dictionary compression methods. Other applications of LPF tables include detecting all runs [17], computing leftmost maximal periodicities [18] and testing square freeness of a string [5].

* This work was supported in part by the National Basic Research Program of China Grant 2007CB807900, 2007CB807901, and the National Natural Science Foundation of China Grant 61033001, 61061130540, 61073174.

Various types of table-like data structures are very useful in designing algorithms on strings, trees or sequences. The Border table is an essential part of the famous KMP algorithm for string-matching [16]. The Suffix table plays an important role in several string algorithms, including Boyer-Moore algorithm [3], Apostolico-Giancarlo algorithm [1,9] and Gusfield's algorithm [14]. The Prefix table, a dual notion of the Suffix table, also has many applications [6]. All these tables are computable in linear time [16,3,6]. Like others, the LPF table of a given string of length n can be computed in time $O(n)$ independent of the alphabet size. Crochemore and Ilie [7] gave two simple algorithms for computing LPF tables using the Suffix Array [19] and the Longest Common Prefix table.

We study in this paper the *reverse engineering problem* on Longest Previous Factor tables. The *reverse LPF problem* is defined as: Given an integer table W, test whether W is the LPF table of some string s, and return such a string if so. Clément, Crochemore and Rindone [4] raised the open question that whether the reverse LPF problem can be solved efficiently. We give a negative answer to this question (with the belief of $P \neq NP$), by proving that the reverse LPF problem is actually NP-hard. Our proof is by a reduction from the classical NP-complete problem 3-SAT. Since the reduction itself is quite complicated, we only show some intuitive ideas in this extended abstract. The technical details will appear in the full version of this paper.

Our result actually exhibits a natural widely-used string-related data structure for which the reverse problem cannot be solved efficiently. Previously this type of reverse problems have been considered by many authors, and they almost all proposed polynomial-time algorithms, or even linear-time algorithms, for the corresponding problems. Franek et. al. [11] and Duval et. al. [10] both gave a linear-time verification of Border tables, for unbounded alphabets and bounded alphabets respectively. Clément, Crochemore and Rindone [4] provided a linear-time algorithm for characterizing Prefix tables. Gawrychowski, Jeż and Jeż [13] proposed a linear-time algorithm for checking whether a given array is the failure function of the Knuth-Morris-Pratt algorithm (see [16]) for some string. Linear-time algorithms for reversing Suffix tables have been proposed by Bannai et al. [2] and Franek and Smyth [12]. The Parameterized Border tables are considered by I et al. [15] and an $O(n^{1.5})$ time algorithm is given. Crochemore et al. studied minimal- and maximal- cover arrays [8] and designed linear-time algorithms for reversing both arrays. To our knowledge, the only known hard reverse problem prior to ours is that of inferring a string from the set of its runs, due to Matsubara, Ishino and Shinohara [20].

Complementing with the intractability result, we also find a class of nontrivial integer tables, namely the tables whose elements are all 0 or 1, on which the reverse LPF problem can be solved in linear time. Our algorithm runs online; that is, assuming that the elements of the input table is fed to the algorithm one by one, our algorithm will maintain a string which is an LPF-string of the current input table, or a special signal indicating that such string does not exist. We also give a simple yet complete characterization of 0-1 LPF tables.

2 Preliminaries

2.1 Notations and Definitions

A string w with alphabet Σ is a sequence $(w[1], w[2], \cdots, w[n])$ with $w[i] \in \Sigma$ for all $1 \leq i \leq n$. Call n the *length* of w, denoted as $|w| = n$. We say $w[i]$ is the i-th *element* of w, or, $w[i]$ is at *position* i in w. Let $w_1 w_2$ be the concatenation of w_1 and w_2. Thus, we can write $(w[1], w[2], \ldots, w[n]) = w[1]w[2] \cdots w[n]$. Denote $w^k = \underbrace{ww \ldots w}_{k \text{ copies of } w}$ for $k \in \mathbb{N}$. A string of the form c^k, where c is a single character, is called a *unary string of* c, or simply a *unary string*. Given a string w, let $w[i, j] = w[i]w[i+1] \cdots w[j]$ denote the *substring* (or alternatively called *factor*) of w starting from position i to j. If w' is a factor of w, we say that w' *appears* in w. If w' starts from position $i' < i$, we say that w' *appears in* w *before position* i. We also say $w[i, j]$ *appears before* $w[i', j']$ if $i < i'$. If uv is a factor of w, we call u a *length-*$|u|$ *predecessor* of v in w, and call v a *length-*$|v|$ *successor* of u in w. A length-1 predecessor (resp. successor) is simply called a *predecessor* (resp. *successor*).

A string W is called an *integer table*, or simply a *table*, if its alphabet is a subset of the set of non-negative integers, i.e. $\Sigma(W) \subseteq \mathbb{N}$. An integer table W is called *LPF-reversible* if there exists a string w satisfying $LPF_w = W$, and such w is called an *LPF-string* of W.

2.2 Simple Properties of LPF Tables

Let w be a string of length n. Trivially $LPF_w[1] = 0$. It is also clear that $LPF_w[i] = 0$ if and only if $w[i]$ is a new character appeared in w (that is, $w[i] \notin \{w[j] \mid 1 \leq j < i\}$), and $LPF_w[i] \leq 1$ (for $i < n$) if and only if the length-2 factor $w[i, i+1]$ does not appear in w before position i. Furthermore, it holds that $LPF_w[i] \leq n - i + 1$ and $LPF_w[i] \geq LPF_w[i-1] - 1$ (we let $LPF_w[0] = 0$). Other important observations include:

– $LPF_w[i] \leq 1$, then $LPF_w[1, i] = LPF_{w[1,i]}$;
– If $LPF_w[i] > 1$, then for every $1 \leq j \leq i$, $LPF_{w[1,i]}[j] = \min(LPF_w[j], i - j + 1)$.

This statement enables us to "extract" a prefix from the whole table, which is very useful in our construction. The proof is omitted due to lack of space.

3 Reduction from 3-SAT

In this section we demonstrate the main theorem of this paper.

Theorem 1. *The problem of deciding whether a given integer table is LPF-reversible is NP-complete.*

This problem is the decision version of the reverse LPF problem, and is in NP since we can verify the validity of a witness (an LPF-string of the given table) in linear time [7]. Thus we only need to present a polynomial-time reduction from an NP-complete problem to it. We choose the classical problem 3-SAT.

Let $\mathcal{F}$ be a 3-CNF formula with variable set $\{X_i \mid 1 \le i \le n\}$ and clause set $\{C_j \mid 1 \le j \le m\}$. W.l.o.g. we assume $n \ge 3$ and $m \ge 3$. We wish to construct an integer table W such that W is LPF-reversible if and only if $\mathcal{F}$ is satisfiable. Our construction of W basically consists of three consecutive sections:

- **Variable Section**. In this part we add segments corresponding to the variables.
- **Assignment Section**. This part contains gadgets for encoding assignments of the variables.
- **Checking Section**. This part is used for checking whether the corresponding assignment satisfies the formula $\mathcal{F}$.

We will carefully design the three sections to ensure that any LPF-string of W corresponds to a satisfying assignment for $\mathcal{F}$, and vice versa.

3.1 Variable Section

In the variable section, we aim to construct $2n + 2$ characteristic segments for the following $2n+2$ literals and Boolean values: $\{X_i \mid 1 \le i \le n\} \cup \{\overline{X_i} \mid 1 \le i \le n\} \cup \{T, F\}$. For the simplicity of expressions, we denote $X_{n+i} = \overline{X_i}$, $X_{2n+1} = T$ and $X_{2n+2} = F$, and temporarily call all of them variables (instead of literals and Boolean values).

Since LPF tables contain only information about lengths of factors, a natural approach is to use different lengths to distinguish between variables. Our target is to use long unary substrings, of distinct characters and lengths, to encode different variables. Since what we are going to construct is the LPF table, it must contain some special structure that can "force" its LPF-string to contain long unary substrings.

To achieve this, we plant some highly overlapping structures in the LPF table. As an example, consider what happens if the LPF table contains three consecutive entries $100, 101, 102$. From the definition, these numbers mean the length of the longest previous factors for these three positions. But we can in fact learn more. We know that the three substrings, starting from these three positions with length $100, 101, 102$ respectively, have a common substring r of length 98 (that starts with the position of entry 102). This tells us that three copies of r have appeared before them. Moreover, these copies must start from distinct position. To see this, w.l.o.g. assume the two copies of r corresponding to 100 and 101 coincide with each other. Then by the definition of LPF tables, there exists a previous factor of length 102 for the entry 100, a contradiction! Thus, we have three copies of the same long string. In order to get our desired unary substring, it suffices to guarantee the existence of two copies of r such that their starting positions only differ by a small distance.

Now comes the specific constructions. We choose $2n+2$ *odd* integers $\{x_i \mid 1 \leq i \leq 2n+2\}$ as follows:

$$(\forall 1 \leq i \leq 2n)\ x_i = 8n + (6i-2)m + 6i^2 - 6i - 9; \tag{2}$$
$$x_{2n+1} = 2(24n^2 + 12mn + 34n + 8m - 9) + 1; \tag{3}$$
$$x_{2n+2} = 2(24n^2 + 12mn + 58n + 14m + 3) + 1. \tag{4}$$

We call x_i the *characteristic length* of variable X_i. Their values given above are carefully chosen so that they have a "safe-but-not-too-long" distance from each other. Now we formally define W_{VS}, the variable section of W. It comprises $2n+2$ substrings each of which encodes one of the variables.

Definition 1. *For every $1 \leq i \leq 2n+2$, let*

$$W_{VS}^{(i)} := (0, 0, x_i{-}1, x_i{-}2, \cdots, 1, 1^{12i-7}, 0, x_i, x_i{+}1, x_i{+}12i{-}5, x_i{+}12i{-}6, \cdots, 1, 1^{6i-1}). \tag{5}$$

(Recall that 1^k means a unary string of 1 of length k.)

Define the variable section W_{VS} *as:*

$$W_{VS} := (0,\ 1,\ W_{VS}^{(1)},\ W_{VS}^{(2)},\ \cdots,\ W_{VS}^{(2n+2)}).$$

The next lemma asserts that any LPF-string of W_{VS} has a "fixed" structure.

Lemma 1. *W_{VS} is LPF-reversible. Moreover, if w_{VS} is its LPF-string, then:*

$$w_{VS} = (a,\ a,\ w_{VS}^{(1)},\ w_{VS}^{(2)},\ \cdots,\ w_{VS}^{(2n+2)}),$$

where, for every $1 \leq i \leq 2n+2$,

$$w_{VS}^{(i)} = (b_i,\ c_i^{x_i},\ u_i,\ d_i,\ c_i^{x_i+2},\ u_i,\ d_i,\ c_i,\ v_i),$$

($w_{VS}^{(i)}$ corresponds to the factor $W_{VS}^{(i)}$), such that:

(a) $\{a\} \cup \{b_j, c_j, d_j \mid 1 \leq j \leq 2n+2\}$ is a set of $6n+7$ distinct characters;
(b) u_i and v_i are strings that end up with the character "a" and do not contain new characters. Moreover, all length-2 factors of u_i are exactly those length-2 strings (with already appeared characters) that do not appear in W_{VS} before (the first occurrence of) u_i; similarly, all length-2 factors of v_i consists of precisely those length-2 strings (with old characters) that do not appear in W_{VS} before v_i.

The rigorous proof is omitted from this short version, and here we just give some intuitions. The crux of the proof is to analyze, as stated before the explicit construction, the common substring r of length $x_i - 2$ resulted from the highly overlapping structures beginning with $(x_i, x_i+1, x_i+12i-5)$. We wish to force the LPF-string to contain long unary substrings ($c_i^{x_i}$ and $c_i^{x_i+2}$ in $w_{VS}^{(i)}$, defined in Lemma 1), which will be regarded as the "encoding" of the corresponding variable. We hope to prove that there are two copies of r whose starting positions differ only by 1. The choice of $\{x_i\}$ is to ensure that x_{i+1} is large enough so that such "long repeated pattern" cannot appear in the previous parts $w_{VS}^{(j)}, j < i$. Another gadget is simply the 0 before $(x_i, x_i + 1, x_i + 12i - 5)$. Since this 0 corresponds to a new character d_i, it ensures that, if all three copies of r start before d_i, then they also end before d_i. To see this, just notice that if one of them contains d_i, so do the other two; but then they must locate at the same position, contradicting the previous analysis. Furthermore, they must end before the 1^{12i-7} part, since an entry "1" in the LPF table means that the length-2 substring starting from it has not appeared before, but there are three copies of r at different positions! Therefore, if all three copies of r start before d_i, then they are restricted in a small range $(0, 0, x_i - 1, x_i - 2, \ldots, 1)$, which will lead to contradictions by simple arguments. Thus, at least one of the three r's appear after d_i. However, since they are "previous factors" of the substring starting from the position of $x_i + 12i - 5$, there are only two possible starting positions for them. Thus, a case-by-case investigation will help us find out the only possibility of their positions, which turns out to be exactly what we want. Finally, the 1^{6i-1} segment at the end of $w_{VS}^{(i)}$ serves as a "clean-up" procedure, which ensures that all the results for $w_{VS}^{(i)}$ similarly apply to $w_{VS}^{(i+1)}$, thus leading to an inductive proof of the lemma.

It should be noticed that this "proof pattern", of utilizing highly overlapping structures and carefully designed gadgets (for example, using new characters as "separators" that cannot be contained in the considered substring) to force the LPF-string to have a "good" form, is the most significant idea of the whole proof.

We end up this section by giving a formal definition of the *characteristic segment* of a variable, which encodes the corresponding variable as a special factor in the LPF-string. This concept will frequently appear in the following sections.

Definition 2. *A factor of W's LPF-string is called a* characteristic segment *of X_i, if it has the form (α, $c_i^{x_i+\delta}$, $u_i[1]$) for some $\delta \geq 0$, where α is a new character that has not appeared before. (Recall that x_i is the characteristic length of X_i defined before, and u_i and c_i are defined in Lemma 1.)*

3.2 Assignment Section

The goal of the assignment section is to associate each variable X_i, $1 \leq i \leq 2n$, with a Boolean value T or F. In this section, we provisionally "forget" the relation between X_i and $X_{n+i} (= \overline{X_i})$, and just regard them as independent variables. The legality of the assignment will be checked in the next part, namely the checking section. Since "assignment" is the theme of this section, the two Boolean values T and F will be highlighted. Thus, for the sake of clearness, we write T and F as the subscripts corresponding to the two values, instead of using $2n + 1$ and $2n + 2$ as before. More formally, for $I \in \{x, b, c, d, u\}$, let I_T and I_F denote I_{2n+1} and I_{2n+2}, respectively. For example, $x_T = x_{2n+1}$ and $d_F = d_{2n+2}$. We also write $T = c_T (= c_{2n+1})$ and $F = c_F (= c_{2n+2})$ for convenience, i.e., we do not distinguish between the two Boolean values and the characters associated with them. (Recall that c_i is the character used for encoding X_i.)

For every variable X_i, we first construct its characteristic segment, and then append to it a "value segment" of length y_i, with the hope that the corresponding part of the LPF-string will be forced to have the form $t_i^{y_i}$ where $t_i \in \{T, F\}$; t_i is then interpreted as the assigned Boolean value to X_i. Once we want to inquire for the value of X_i, we construct a special gadget to extract t_i. Due to some technique issues, the lengths of the value segments $\{y_i \mid 1 \leq i \leq 2n\}$ must be carefully selected. We choose them as:

$$(\forall 1 \leq i \leq 2n)\; y_i = 24n^2 + 12mn + 28n + 8m - 14 + 3i. \tag{6}$$

We need some other definitions to make the statement below clearer. Given a string w and one of its unary substring s (recall that a unary string is one that has the form c^k), we say u is a *non-trivial predecessor* (resp. *non-trivial successor*) of s in w if u is a predecessor (resp. successor) of the maximal unary substring of w that contains s as a factor. For example, in the string *abcaaaade*, we say c is a non-trivial predecessor of aa, and d is a nontrivial successor of aaa.

We now define the assignment section of W.

Definition 3. *For every* $1 \leq i \leq 2n$, *let*

$$W_{AS}^{(i)} = (0,\ x_i{+}2,\ x_i{+}3,\ x_i{+}3,\ x_i{+}2,\ \cdots,\ 1,\ 0,\ y_i{+}1,\ y_i{+}1,\ y_i,\ y_i{-}1,\ \ldots,\ 1).$$

Define the assignment section W_{AS} *as*

$$W_{AS} := (W_{AS}^{(1)},\ W_{AS}^{(2)},\ \cdots,\ W_{AS}^{(2n)}).$$

Lemma 2. *Let W_{VS} and W_{AS} be defined as in Definitions 1 and 3. Then the string $W_{VS}W_{AS}$ is LPF-reversible. Furthermore, suppose $w_{VS}w_{AS}$ is an LPF-string of it with $|w_{VS}| = |W_{VS}|$ (and hence $|w_{AS}| = |W_{AS}|$). Then, w_{VS} has the same form as stated in Lemma 1, and*

$$w_{AS} = (w_{AS}^{(1)},\ w_{AS}^{(2)},\ \cdots,\ w_{AS}^{(2n)}),$$

where for every $1 \leq i \leq 2n$,

$$w_{AS}^{(i)} = (e_i,\ c_i^{x_i+4},\ u_i[1],\ f_i,\ r_i,\ t_i^{y_i},\ s_i),$$

($w_{AS}^{(i)}$ corresponds to the factor $W_{AS}^{(i)}$), such that:

(a) e_i, f_i are new characters never appeared before;
(b) $t_i \in \{T, F\}$;
(c) r_i and s_i are respectively non-trivial predecessor and successor of $t_i^{y_i}$ in w_{VS}.

The key point of the proof of Lemma 2 is to show that the substring $(0,\ x_i + 2,\ x_i + 3,\ x_i + 3,\ x_i + 2,\ \cdots,\ 1,\ 0)$ of $W_{AS}^{(i)}$ must correspond to $e_i c_i^{x_i+4} u_i[1]$ (a characteristic segment of X_i) in its LPF-string. This special structure enables us to "extract" c_i, the encoding of the variable X_i, which is very useful since we need to deal with the variables everywhere in the whole reduction. The proof is omitted from this short version.

Up till now, we have successfully associated each variable X_i with a Boolean value t_i by appending a gadget string, which is forced to have the form $t_i^{y_i}$, to the characteristic segment of X_i. However, it is possible that both X_i and $\overline{X_i}$ are assigned with the same value, resulting in an infeasible assignment. We will deal with this issue in the checking section.

3.3 Checking Section

The preceding parts of our construction involved no logical connections between variables. The assignment section can encode any possible assignment to $\{X_i \mid 1 \le i \le 2n\}$, including those invalid ones (i.e. $X_i = X_{n+i}$ for some i). Moreover, $W_{VS}W_{AS}$ is always LPF-reversible regardless of whether $\mathcal{F}$ is satisfiable or not. The checking section is designed to handle these problems. Recall that $\{t_i \mid 1 \le i \le 2n\}$ is an assignment to $\{X_i \mid 1 \le i \le 2n\}$ induced by the assignment section W_{AS}. The checking section W_{CS} consists of 3 consecutive substrings $W_{CS1}, W_{CS2}, W_{CS3}$. We use W_{CS1} (resp. W_{CS2}) to guarantee that W is LPF-reversible implies at least one of t_i and t_{n+i} is F (resp. T). Hence, the first two parts enforce the assignment to be valid. The last part W_{CS3} is to ensure that for every clause C_j, at least one of its literals is assigned with T, if W is LPF-reversible. On the other hand, it is fairly easy to prove that the satisfiability of $\mathcal{F}$ implies the LPF-reversibility of W (just follow the LPF-strings defined in the lemmas). Thus, combining three parts together assures us that W is LPF-reversible if and only if $\mathcal{F}$ is satisfiable.

In fact, the constructions of the three parts have similar structures; their tasks are all to check if there is a certain value (T or F) among the assigned values of several (in fact 2 or 3) variables. Therefore, in this extended abstract we only provide the definition and results of W_{CS1}. The constructions of W_{CS2} and W_{CS3}, as well as the rigorous proofs, will appear in the full paper.

We shall briefly explain the (somewhat seemingly unnatural) technique used to guarantee at least one pre-specified value (say T) among some variables. We construct some gadgets in such a way that each of them implies the existence of a distinct, previously appeared character-pair (p, q), where p and q are, respectively, non-trivial predecessor and successor of a long unary string of T or F (that is, T^z or F^z for some large integer z). Thus, some number of gadgets imply the same number of such non-trivial (predecessor, successor)-pairs of T^z or F^z. We can actually calculate the number of such pairs in our construction till now. We then create some new successors by appending new characters to long unary strings of the values that we need to check; note that we don't know whether the values are T or F, but we want at least one of them to be T. By some careful design, we make sure that there are enough number of such pairs if and only if at least one of the new successors is a successor of T^z; in other words, at least one of the values we consider is T. Some further discussions are given after the statement of Lemma 3.

We now formally define the first part of our checking section.

Definition 4. *For each $i \in \{1, 2, \ldots, n\}$, define*

$$\begin{aligned} W_{CS1}^{(i)} := \; & (0,\ x_i + 4,\ x_i + 5,\ x_i + 8 + z_i,\ x_i + 7 + z_i,\ \ldots,\ 1,\ 0, \\ & 0,\ x_{n+i} + 4,\ x_{n+i} + 5,\ x_{n+i} + 8 + z_i,\ x_{n+i} + 7 + z_i,\ \ldots,\ 1,\ 0, \\ & (0,\ z_i + 1,\ z_i + 1,\ z_i,\ \cdots,\ 1)^{10}). \end{aligned}$$

Define the first part of the checking section as:

$$\begin{aligned} W_{CS1} := \; & (0,\ 0,\ x_F + 2,\ x_F + 3,\ x_F + 3,\ x_F + 2,\ \ldots,\ 1, \\ & W_{CS1}^{(1)},\ W_{CS1}^{(2)},\ \cdots,\ W_{CS1}^{(n)}). \end{aligned}$$

The substring $W_{CS}^{(i)}$ is also called a *block*, which is the minimum unit of a complete verification of the assignment for one variable. $W_{CS}^{(i)}$ is to check that at least one of X_i and $X_{n+i} (= \overline{X_i})$ is assigned with F.

To avoid the circumstances where successors introduced by former blocks interfere with the following blocks, we need to choose $\{z_i \mid 1 \le i \le 2n+m\}$ with enough distances as follows: (Here $\{z_i \mid n+1 \le i \le 2n\}$ is used for the second part of checking section, and $\{z_i \mid 2n+1 \le i \le 2n+m\}$ is used for the third one; recall that the 3-CNF formula $\mathcal{F}$ has m clauses.)

$$(\forall 1 \le i \le 2n+m)\ z_i = 24n^2 + 12mn + 20n + 4m - 13 + 4i.$$

Lemma 3. *If $W_{VS}W_{AS}W_{CS1}$ is LPF-reversible, then, for any LPF-string w of it, $w = w_{VS}w_{AS}w_{CS1}$ holds, where w_{VS} and w_{AS} are defined as in Lemmas 1 and 2, and w_{CS1} has the following form:*

$$w_{CS1} = (g_F,\ r_F,\ F^{x_F+4},\ u_F[1], w_{CS1}^{(1)},\ w_{CS1}^{(2)}, \ldots,\ w_{CS1}^{(n)}),$$

where, for every $1 \le i \le n$,

$$\begin{aligned} w_{CS1}^{(i)} = \; & (g_i,\ c_i^{x_i+6},\ u_i[1],\ f_i,\ r_i,\ t_i^{z_i+1},\ h_i, \\ & g_{n+i},\ c_{n+i}^{x_{n+i}+6},\ u_{n+i}[1],\ f_{n+i},\ r_{n+i},\ t_{n+i}^{z_i+1},\ h_{n+i}, \\ & g_{i,1},\ r_{i,1},\ t_{i,1}^{z_i},\ s_{i,1}, \\ & \vdots \\ & g_{i,10},\ r_{i,10},\ t_{i,10}^{z_i},\ s_{i,10}), \end{aligned}$$

($w_{CS1}^{(i)}$ corresponds to the factor $W_{CS1}^{(i)}$), such that:

(a) $\{g_i, h_i \mid 1 \le i \le 2n\} \cup \{g_{i,j} \mid 1 \le i \le n, 1 \le j \le 10\} \cup \{g_F, r_F\}$ is a set of distinct new characters;
(b) $\forall 1 \le i \le n, \forall 1 \le j \le 10$, $t_{i,j} \in \{T, F\}$ and $r_{i,j}, s_{i,j}$ are previously appeared non-trivial predecessor and successor of $t_{i,j}^{z_i}$;
(c) $\forall 1 \le i \le n$, at least one of t_i and t_{n+i} must be F.

Conversely, if there exists a string $w = w_{VS}w_{AS}w_{CS1}$ of the above form, then $W_{VS}W_{AS}W_{CS1}$ is LPF-reversible and w is its LPF-string.

An important step in proving Lemma 3 is to show that any substring $(0,\ x_i + 4,\ x_i + 5,\ x_i + 8 + z_i,\ x_i + 7 + z_i,\ \ldots,\ 1,\ 0)$ of $W_{CS1}^{(i)}$ must correspond to $(g_i, c_i^{x_i+6}, u_i[1], f_i, r_i, t_i^{z_i+1}, h_i)$ in its LPF-string. This allows us to extract c_i as well as t_i, the assigned value of the variable X_i, which is important for later use.

We now give some intuitive ideas of how such scheme works. As claimed before, the values of X_i and $\overline{X_i}$ (that is, t_i and t_{n+i}) can be extracted out, and h_i, h_{n+i} correspond to non-trivial successors of $t_i^{z_i}$ and $t_{n+i}^{z_i}$, respectively. With similar analysis to the proof of Lemma 2, every component $(0,\ z_i+1,\ z_i+1,\ z_i,\ \cdots,\ 1)$ in $W_{CS1}^{(i)}$ corresponds to $g_{i,j}r_{i,j}t_{i,j}^{z_i}s_{i,j}$ in the LPF-string. Thus, $(r_{i,j}, s_{i,j})$ is a non-trivial (predecessor, successor)-pair of T^{z_i} or F^{z_i} that appeared before. We can also argue that these 10 pairs should be pairwise different, which implies the existence of 10 different non-trivial (predecessor, successor)-pairs of T^{z_i} or F^{z_i}. We calculate the number of such predecessors and successors that have appeared so far: T^{z_i} has b_T, d_T as its predecessors, and $u_T[1]$ as a successor; F^{z_i} has predecessors b_F, d_F, r_F and a successor $u_F[1]$; also, there are two successors h_i and h_{n+i} (of $t_i^{z_i}$ and $t_{n+i}^{z_i}$, respectively) yet to be determined whose successor they actually are. On the one hand, if both h_i, h_{n+i} are successors of T^{z_i}, there are $2 \times 3 + 3 \times 1 = 9$ (predecessor, successor)-pairs in all, contradicting the requirement of 10 pairs. On the other hand, it is easy to verify the existence of 10 or 11 such pairs when at least one of h_i and h_{n+i} is a successor of F^{z_i}. Thus, when $W_{VS}W_{AS}W_{CS1}$ is LPF-reversible, at least one of h_i and h_{n+i} should be a successor of F^{z_i}, implying that at least one of t_i and t_{n+i} should be F. The converse direction is only a matter of tedious verification.

Applying similar techniques to construct W_{CS2} and W_{CS3} completes our reduction. The details of the constructions of W_{CS2} and W_{CS3} are omitted due to lack of space, and will appear in the full paper. The final (minor) step for proving Theorem 1 is to show that the reduction runs in polynomial time, which directly follows from our construction. We can also strengthen Theorem 1 as follows by a padding argument.

Theorem 2. *For every constant $\epsilon > 0$, it is NP-complete to decide whether a table of length n with at most n^ϵ zeros is LPF-reversible.*

Note that we need an unbounded size alphabet in our hardness proof. It is interesting to see what happens if the alphabet size is bounded. Formally, we propose the following open question.

Question 1. Is it NP-complete to decide whether a table that contains at most k zeros is LPF-reversible, where k is a fixed integer?

4 LPF Tables with 0-1 Entries

An integer table whose elements are all 0 or 1 is called a *0-1 table.* In this section, we prove that the reverse LPF problem is solvable in linear time over

0-1 tables. Moreover, we give a complete characterization of LPF-reversible 0-1 tables (Theorem 3). Given a table W and an integer i, define

$$Num(W, i) := |\{j \mid W[j] = i, 1 \leq j \leq |W|\}|,$$

which is the number of elements in W that are equal to i.

Theorem 3. *A 0-1 table W of length n is LPF-reversible if and only if*

$$W[1] = 0 \text{ and } (\forall 2 \leq i \leq n)\ i \leq (Num(W[1, i], 0))^2 + 1 . \quad (7)$$

Theorem 4. *There is a linear-time online algorithm for the reverse LPF problem on 0-1 tables.*

It is easy to see that Theorem 3 implies the decision part of Theorem 4, since we can test whether (7) holds or not in linear time simply using two counters. The construction of an LPF-string in linear time needs more effort.

The "only if" direction of Theorem 3 is easy. Suppose the input 0-1 table W of length n is LPF-reversible and let w be an LPF-string of W. Since $W[i] \leq 1$ for each i, we know from Section 2.2 that for every $i \in \{1, 2, \ldots, n-1\}$, the length-2 factor $W[i, i+1]$ does not appear in t before position i. Thus, $w[1, i]$ contains $i-1$ distinct length-2 factors, whereas the total number of such factors is $(Num(T[1, i], 0))^2$. We therefore have $i - 1 \leq (Num(W[1, i], 0))^2$ for each $2 \leq i \leq n$. Together with the trivial fact that $W[1] = 0$, the "only if" part of Theorem 3 follows.

The idea of proving the "if" direction is to show the equivalence between the LPF-reversibility of W and the existence of a "partial" Eulerian path with some constraints in a directed graph associated with W. Let $ZeroW = \{i \mid W[i] = 0, 1 \leq i \leq n\}$ and $m = |ZeroW|$. Assume $ZeroW = \{i_1, i_2, \ldots, i_m\}$ where $1 = i_1 < i_2 < \ldots < i_m$ (by Equation (7) we have $1 \in ZeroW$). Let G be a complete directed graph on the vertex set $V = \{1, 2, \ldots, m\}$ with all self-loops included. A (not necessarily simple) path in G is called a *partial Eulerian path* if it traverses each edge at most once. We can prove that the following two statements are equivalent:

1. W is LPF-reversible.
2. G has a partial Eulerian path P of length $n-1$ (thus it contains n vertices), such that for every $1 \leq j \leq m$, the i_j-th vertex on P is j, and it has not been visited by P before.

The linear time online algorithm for the reverse LPF problem on 0-1 tables is just based on a linear time algorithm for finding such a partial Eulerian path. The detailed proofs of Theorems 3 and 4 will appear in the full version of this paper.

Acknowledgements

We would like to thank Xiaoming Sun for helpful discussions, and the anonymous referees for their suggestions on improving the presentation of this paper.

References

1. Apostolico, A., Giancarlo, R.: The Boyer-Moore-Galil string searching strategies revisited. SIAM J. Comput. 15(1), 98–105 (1986)
2. Bannai, H., Inenaga, S., Shinohara, A., Takeda, M.: Inferring strings from graphs and arrays. In: Rovan, B., Vojtáš, P. (eds.) MFCS 2003. LNCS, vol. 2747, pp. 208–217. Springer, Heidelberg (2003)
3. Boyer, R.S., Moore, J.S.: A fast string searching algorithm. Commin. ACM 20(10), 762–772 (1977)
4. Clément, J., Crochemore, M., Rindone, G.: Reverse engineering prefix tables. In: STACS 2009, Freiburg, pp. 289–300 (2009)
5. Crochemore, M.: Transducers and repetitions. Theoret. Comput. Sci. 45(1), 63–86 (1986)
6. Crochemore, M., Hancart, C., Lecroq, T.: Algorithms on strings. Cambridge University Press, Cambridge (2007)
7. Crochemore, M., Ilie, L.: Computing Longest Previous Factor in linear time and applications. Inf. Process. Lett. 106(2), 75–80 (2008)
8. Crochemore, M., Iliopoulos, C.S., Pissis, S.P., Tischler, G.: Cover Array string reconstruction. In: Amir, A., Parida, L. (eds.) CPM 2010. LNCS, vol. 6129, pp. 251–259. Springer, Heidelberg (2010)
9. Crochemore, M., Lecroq, T.: Tight bounds on the complexity of the Apostolico-Giancarlo algorithm. Inf. Process. Lett. 63(4), 195–203 (1997)
10. Duval, J.-P., Lecroq, T., Lefebvre, A.: Efficient validation and construction of border arrays. In: Proceedings of 11th Mons Days of Theoretical Computer Science, Rennes, France, pp. 179–189 (2006)
11. Franek, F., Gao, S., Lu, W., Ryan, P.J., Smyth, W.F., Sun, Y., Yang, L.: Verifying a Border array in linear time. J. Combinatorial Math. and Combinatorial Computing 42, 223–236 (2002)
12. Franek, F., Smyth, W.F.: Reconstructing a Suffix Array. International Journal of Foundations of Computer Science 17(6), 1281–1295 (2006)
13. Gawrychowski, P., Jeż, A., Jeż, Ł.: Validating the Knuth-Morris-Pratt failure function, fast and online. In: Ablayev, F., Mayr, E.W. (eds.) CSR 2010. LNCS, vol. 6072, pp. 132–143. Springer, Heidelberg (2010)
14. Gusfield, D.: Algorithms on strings, trees and sequences: computer science and computational biology. Cambridge University Press, Cambridge (1997)
15. I., T., Inenaga, S., Bannai, H., Takeda, M.: Verifying a Parameterized Border Array in $O(n^{1.5})$ time. In: Amir, A., Parida, L. (eds.) CPM 2010. LNCS, vol. 6129, pp. 238–250. Springer, Heidelberg (2010)
16. Knuth, D.E., Morris, J.H., Pratt, V.R.: Fast pattern matching in strings. SIAM J. Comput. 6(1), 323–350 (1977)
17. Kolpakov, R., Kucherov, G.: Finding maximal repetitions in a word in linear time. In: FOCS 1999, pp. 596–604. IEEE Computer Society Press, New York (1999)
18. Main, M.G.: Detecting leftmost maximal periodicities. Discrete Applied Math. 25, 145–153 (1989)
19. Manber, U., Myers, G.: Suffix arrays: a new method for on-line search. SIAM J. Comput. 22(5), 935–948 (1993)
20. Matsubara, W., Ishino, A., Shinohara, A.: Inferring strings from runs. In: Prague Stringology Conference 2010, pp. 150–160 (2010)
21. Ziv, J., Lempel, A.: A Universal algorithm for sequential data compression. IEEE Trans. Inform. Theory 23, 337–342 (1977)
22. Ziv, J., Lempel, A.: Compression of individual sequences via variable-rate coding. IEEE Trans. Inform. Theory 24, 530–536 (1978)