Information Processing Letters 168 (2021) 106097

Contents lists available at ScienceDirect

Information Processing Letters

www.elsevier.com/locate/ipl

Longest previous overlapping factor array

Hideo Bannai [a], Shunsuke Inenaga [b,c], Neerja Mhaskar [d,*]

[a] M&D Data Science Center, Tokyo Medical and Dental University, Japan
[b] Department of Informatics, Kyushu University, Japan
[c] PRESTO, Japan Science and Technology Agency, Japan
[d] Department of Computing and Software, McMaster University, Canada

ARTICLE INFO

Article history:
Received 31 December 2019
Received in revised form 18 November 2020
Accepted 13 January 2021
Available online 26 January 2021
Communicated by Abhi Shelat

Keywords:
Algorithms
Longest previous overlapping factor
Overlapping factor
Manhattan skyline problem

ABSTRACT

In this paper, we introduce the *longest previous overlapping factor array* of a string – a variant of the longest previous factor array. We show that it can be computed in linear time in the length of the input string, via a reduction to the Max-variant of the Manhattan skyline problem Crochemore et al. (2014) [5].

1. Introduction

Crochemore and Ilie in [3] introduced the *longest previous factor array* (LPF): given a string $\boldsymbol{w}$ of length n, for any position i in $\boldsymbol{w}$, $\texttt{LPF}[i]$ stores the length of the longest prefix of $\boldsymbol{w}[i..n]$ that has a previous occurrence which begins in the position range $[1..i-1]$ of $\boldsymbol{w}$. Several linear time algorithms have been proposed to compute the LPF array [3,4]. Of the many applications of the LPF array, one of its most important application is its use in the computation of the famous Lempel-Ziv factorization [10] used for data compression.

Later, an important variant of the LPF array, called the *longest previous non-overlapping factor array* (LPnF) was proposed [6,7]: $\texttt{LPnF}[i]$ stores the length of the longest prefix of $\boldsymbol{w}[i..n]$ that has a previous occurrence which ends in position range $[1..i-1]$ of $\boldsymbol{w}$. Thus, for any $1 \leq i \leq n$, the corresponding factor $\boldsymbol{w}[i..i+\texttt{LPnF}[i]-1]$ must have a previous occurrence that *does not* overlap with position i. It is known that the LPnF array of a string of length n can be computed in $\mathcal{O}(n)$ time [6,7]. The main application of LPnF is the Lempel-Ziv factorization *without self-references* (a.k.a. *f-factorization*) [2], which plays a central role in the solutions to various string problems such as approximation of the smallest grammar-based compression [9], and computation of all maximal repetitions (a.k.a. runs) [8] in a given string, just to mention a few. Other variants of the LPF array have been proposed in the literature, for example the longest reverse factor array [6], which can also be computed in linear time in the length of the input string.

In this paper, we discuss another variant of LPF, called the *longest previous overlapping factor array* (LPoF) of a given string. Namely, $\texttt{LPoF}[i]$ stores the length of the longest prefix of $\boldsymbol{w}[i..n]$ that has a previous occurrence which ends in the position range $[i..i+\texttt{LPoF}[i]-2]$ of $\boldsymbol{w}$. Thus, for any $1 \leq i \leq n$, the corresponding factor $\boldsymbol{w}[i..i+\texttt{LPnF}[i]-1]$ must have a previous occurrence that *does* overlap with position i. The LPoF array is a natural extension to the LPF array and can be seen as a complement to LPnF, since $\texttt{LPF}[i] = \max\{\texttt{LPnF}[i], \texttt{LPoF}[i]\}$ always holds. While there exists a simple $\mathcal{O}(n)$-time algo-

* Corresponding author.
E-mail addresses: hdbn.dsc@tmd.ac.jp (H. Bannai), inenaga@inf.kyushu-u.ac.jp (S. Inenaga), pophlin@mcmaster.ca (N. Mhaskar).

https://doi.org/10.1016/j.ipl.2021.106097
0020-0190/

i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
$\boldsymbol{w}$	a	b	a	a	a	b	a	b	a	a	a	b	a	b	a	a	b
LPF	0	0	1	2	3	2	10	9	8	7	6	5	4	3	3	2	1
LPnF	0	0	1	1	3	2	6	6	6	6	6	5	4	3	3	2	1
LPoF	0	0	0	2	0	0	10	9	8	7	0	0	3	0	0	0	0

Fig. 1. The LPF, LPnF, and LPoF arrays for string $\boldsymbol{w} =$ abaaababaaabaabaab.

rithm which computes the LPF array from the *Suffix Array* and the *Longest Common Prefix Array* [3,4], the above equation suggests yet another approach for computing the LPF array.

On the other hand, we remark that the LPoF array cannot be computed trivially from either or both the LPF and LPnF arrays. This poses an interesting theoretical question: Can we compute the LPoF array in $\mathcal{O}(n)$ time? We answer this question affirmatively, by presenting a linear-time algorithm which computes the LPoF array using a linear-time algorithm for solving a variant of the Manhattan skyline problem [5].

The outline of the paper is as follows: in Section 2, we give the mathematical background and briefly discuss a variant of the Manhattan skyline problem. In Section 3, we present our algorithm to compute the LPoF array.

2. Preliminaries

2.1. Strings

A string $\boldsymbol{w}$ is an ordered sequence of *letters* or *symbols* chosen from a finite totally ordered set of Σ, called an *alphabet*. The *length* of $\boldsymbol{w}$ is written $|\boldsymbol{w}| = n$. Throughout this paper, we assume that Σ is the integer alphabet of polynomial size in n.

We also write a string $\boldsymbol{w}$ of length n in an array form as $\boldsymbol{w}[1..n]$. We denote by $\boldsymbol{w}[i]$ the i-th letter of $\boldsymbol{w}$. A string $\boldsymbol{w}[i..j]$, where $1 \le i \le j \le n$, is called a *substring* or a *factor* of $\boldsymbol{w}[1..n]$. A substring $\boldsymbol{w}[i..j]$ is a *proper substring* of $\boldsymbol{w}$ if $j - i + 1 < n$. A substring $\boldsymbol{w}[1..j]$, where $1 \le j \le n$ is called a *prefix* of $\boldsymbol{w}$. A substring $\boldsymbol{w}[i..n]$, where $1 \le i \le n$ is called a *suffix* of $\boldsymbol{w}$.

A positive integer p is said to be a *period* of a string $\boldsymbol{w}$ of length n if $\boldsymbol{w}[i] = \boldsymbol{w}[i + p]$ for any $1 \le i \le n - p$. A non-empty string $\boldsymbol{w}$ of even length is said to be a *square* if $\boldsymbol{w}$ has a period $\frac{|\boldsymbol{w}|}{2}$ (or equivalently if $\boldsymbol{w}$ is of the form $\boldsymbol{xx}$). A substring $\boldsymbol{u} = \boldsymbol{w}[i..j]$ of $\boldsymbol{w}$ is called a *repetition* in $\boldsymbol{w}$ if the smallest period p of $\boldsymbol{u}$ satisfies $p \le \frac{|\boldsymbol{u}|}{2} = \frac{j-i+1}{2}$. A repetition $\boldsymbol{u} = \boldsymbol{w}[i..j]$ of $\boldsymbol{w}$ is said to be a *maximal repetition* or a *run* in $\boldsymbol{w}$ if the period p cannot be extended to the left or to the right, i.e., $i = 1$ or $\boldsymbol{w}[i-1] \neq \boldsymbol{w}[i+p-1]$, and $j = n$ or $\boldsymbol{w}[j+1] \neq \boldsymbol{w}[j-p+1]$. Each run $\boldsymbol{r}$ in $\boldsymbol{w}$ is represented by a triple (i, j, p) such that $\boldsymbol{r} = \boldsymbol{w}[i..j]$ and p is the smallest period of $\boldsymbol{r}$.

Theorem 1 *([8,1]). For any string $\boldsymbol{w}$ of length n, the number of runs in w is $\mathcal{O}(n)$. Also, all runs in $\boldsymbol{w}$ can be computed in $\mathcal{O}(n)$ time and space if $\boldsymbol{w}$ is drawn from an integer alphabet of polynomial size in n.*

The paper [1] presents a new and much simpler algorithm for calculating runs.

2.2. Longest previous overlapping factor array

Given a string $\boldsymbol{u}$ of the form $\boldsymbol{u} = \boldsymbol{vx} = \boldsymbol{yv}$, where $|\boldsymbol{u}| > |\boldsymbol{v}| > |\boldsymbol{x}|$, $\boldsymbol{v}$ is called an *overlapping factor*. For example, given the string $\boldsymbol{u} =$ abacabacaba $=$ $(\text{abac})^2$aba, abacaba is an overlapping factor of $\boldsymbol{u}$.

Given a string $\boldsymbol{w}$ of length n, the *Longest Previous Overlapping Factor array* (LPoF) is an integer array of length n where each element

$$\text{LPoF}[i] = \max\{\ell \ge 2 \mid \boldsymbol{w}[k..k+\ell-1] = \boldsymbol{w}[i..i+\ell-1], \\ k < i \le k+\ell-1\}$$

if such ℓ exists, otherwise it is 0.

Fig. 1 gives the LPF, LPnF, and LPoF arrays corresponding to the string $\boldsymbol{w} =$ abaaababaaabaabaab. It is easy to see that the LPF array can be trivially computed from both the LPnF and LPoF arrays, as the i-th element of the LPF array is simply the maximum of the i-th elements of the LPnF and LPoF arrays; that is, $\text{LPF}[i] = \max\{\text{LPnF}[i], \text{LPoF}[i]\}$ for any $1 \le i \le n$.

Note however that the LPF array can also be computed in linear time using the Suffix Array and the Longest Common Prefix Array, which is conceptually simpler than the approach mentioned above.

2.3. Manhattan skyline problem

Crochemore et al. [5] presented the Min-variant of the Manhattan Skyline Problem, and showed that it can be computed in linear time. Here we consider the Max-variant of the Manhattan Skyline Problem (MSP) as shown below.

Input: A set $\mathcal{S}$ of $\mathcal{O}(n)$ subintervals of $[1..n]$, where each subinterval $[i..j]$ is associated with a natural weight, $\mathit{height}([i..j])$, of size $\mathcal{O}(n)$.

Output: A table $f[t] = \max\{\mathit{height}([i..j]) \mid t \in [i..j], [i..j] \in \mathcal{S}\}$, $t \in [1..n]$.

The Max-variant of MSP can be solved in $\mathcal{O}(n)$ time using the same strategy given in [5] to solve the Min-variant of MSP with the exception of sorting the intervals in non-increasing order instead of non-decreasing order. Consequently, we obtain the following lemma.

Lemma 2. *The Max-variant of* MSP *can be solved in $\mathcal{O}(n)$ time.*

3. Computing the LPoF array

We follow a similar schema presented in [5] for computing internal local periods of a string, to compute the

LPoF array values; that is, we infer the LPoF array values from the run structure of $\boldsymbol{w}$. More specifically, we observe that any overlapping string in $\boldsymbol{w}$ corresponds to one of the runs in $\boldsymbol{w}$. If $\texttt{LPoF}[i] = \ell$, then for some k such that $\ell > i - k$, $\boldsymbol{w}[k..i-1] = \boldsymbol{w}[i..i + (i-k) - 1]$, and $\boldsymbol{w}[k..i + (i-k) - 1]$ is a square of period $i - k$. This also implies that there must exist a run (b, e, p) with period $p = i - k$ containing position i. Furthermore, it must satisfy $b + p \leq i \leq e - p$. The last condition is due to the fact that a run (b, e, p) with $\min\{1, i - p + 1\} \leq b$ or $e \leq \min\{i + p - 1, n\}$ does not contribute to the value of $\texttt{LPoF}[i]$. For example in Fig. 1, although the run $(1, 16, 6)$ contains the position $i = 11$, it does not satisfy the condition $b + p \leq i \leq e - p$, and therefore it does not contribute to the value of LPoF[11].

The rationale for the last condition is as follows. For two occurrences of a substring $\boldsymbol{u}$, where $|\boldsymbol{u}| = \ell$, to overlap in a run, ℓ must be larger than the period p of the run. Consequently, these overlapping occurrences, say at indices i_1 and $i_2 > i_1$, are separated by p positions in the run; that is, $i_2 - i_1 = p$. However, for a substring occurring at an index $b \leq i < b + p$, its previous overlapping occurrence must be at $i - p < b$. This case is not possible as the run would be left extensible. Further, for a substring starting at index $e - p < i \leq e$, it has a length $< p$ in the run, and as result does not overlap with any of its occurrences in the run. Therefore, both these cases do not contribute to the $\texttt{LPoF}[i]$ values.

There can be more than one such run that contains position i, but the one that gives the largest ℓ is the one with the largest endpoint e. For example, in Fig. 1 the position $i = 7$ is contained in two runs $(1, 16, 6)$, and $(5, 9, 2)$ which satisfy the above conditions. Since we are interested in the largest ℓ, we consider the run $(1, 16, 6)$ with the largest endpoint $e = 16$.

To this end, we reduce the computation of each element of the LPoF array to an instance of the Max-variant of the Manhattan Skyline problem as follows:

$$\texttt{LPoF}[i] = \max\{\mathit{height}(t) \mid i \in t, t \in I\} - i + 1,$$

where I is the set of intervals corresponding to the runs in $\boldsymbol{w}$ such that

$$I = \{[b + p..e - p] \mid (b, e, p) \text{ is a run in } \boldsymbol{w}, \text{ and } b + p \leq e - p\},$$

and $\mathit{height}(t) = e$ is the weight of the interval $t = [b + p..e - p]$ for each run (b, e, p). Then, for any i, $\texttt{LPoF}[i]$ is the largest weight of all the intervals containing i, if such intervals exist; otherwise $\texttt{LPoF}[i] = 0$. This is an instance of the Max-variant of the Manhattan Skyline Problem shown above. By Lemma 2, it can be computed in $\mathcal{O}(n)$ time. Since we can compute all runs in $\boldsymbol{w}$ in $\mathcal{O}(n)$ time by Theorem 1, LPoF array can be computed in $\mathcal{O}(n)$ time. Therefore, we have shown the following:

Theorem 3. *For any string $\boldsymbol{w}$ of length n, over an integer alphabet of polynomial size in n, the* LPoF *array of $\boldsymbol{w}$ can be computed in $\mathcal{O}(n)$ time and space.*

For example, the string $\boldsymbol{w}$ given in Fig. 1 contains the following runs: $(1, 16, 6)$, $(3, 5, 1)$, $(5, 9, 2)$, $(9, 11, 1)$, $(11, 15, 2)$ and $(15, 16, 1)$. Then we get,

$$I = \{[7..10], [4..4], [7..7], [10..10], [13..13]\}$$

with weights 16, 5, 9, 11, and 15, respectively. Consider an index $i = 7$. This index is contained in intervals [7..10] and [7..7] with weights 16 and 9 respectively. Therefore, $\texttt{LPoF}[7] = 16 - 7 + 1 = 10$.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgements

This work is based on the authors' discussions during the StringMasters workshop held in Prague, August 29-31, 2018.

This work was supported by JSPS KAKENHI [Grant Numbers JP16H02783, 20H04141 (HB), JP17H01697 (SI)], and by JST PRESTO [Grant Number JPMJPR1922 (SI)].

References

[1] H. Bannai, T. I, S. Inenaga, Y. Nakashima, M. Takeda, K. Tsuruta, The "Runs" theorem, SIAM J. Comput. 46 (5) (2017) 1501–1514.
[2] M. Crochemore, C. Hancart, T. Lecroq, Algorithms on Strings, Cambridge University Press, 2007.
[3] M. Crochemore, L. Ilie, Computing longest previous factor in linear time and applications, Inf. Process. Lett. 106 (2008) 75–80.
[4] M. Crochemore, L. Ilie, C.S. Iliopoulos, M. Kubica, W. Rytter, T. Waleń, Computing the longest previous factor, Electron. J. Comb. 34 (2013) 15–26.
[5] M. Crochemore, C.S. Iliopoulos, M. Kubica, J. Radoszewski, W. Rytter, T. Waleń, Extracting powers and periods in a word from its runs structure, Theor. Comput. Sci. 521 (2014) 29–41.
[6] M. Crochemore, C.S. Iliopoulos, M. Kubica, W. Rytter, T. Waleń, Efficient algorithms for three variants of the LPF table, J. Discret. Algorithms 11 (2012) 51–61.
[7] M. Crochemore, G. Tischler, Computing longest previous non-overlapping factors, Inf. Process. Lett. 111 (6) (2011) 291–295.
[8] R.M. Kolpakov, G. Kucherov, Finding maximal repetitions in a word in linear time, in: FOCS 1999, 1999, pp. 596–604.
[9] W. Rytter, Application of Lempel-Ziv factorization to the approximation of grammar-based compression, Theor. Comput. Sci. 302 (1–3) (2003) 211–222.
[10] J. Ziv, A. Lempel, A universal algorithm for sequential data compression, IEEE Trans. Inf. Theory IT-23 (3) (1977) 337–349.