Enhanced string factoring from alphabet orderings

Clare, Amanda; Daykin, Jacqueline W.

Published in:
Information Processing Letters

DOI:
10.1016/j.ipl.2018.10.011

Publication date:
2019

Citation for published version (APA):
Clare, A., & Daykin, J. W. (2019). Enhanced string factoring from alphabet orderings. *Information Processing Letters*, *143*, 4-7. https://doi.org/10.1016/j.ipl.2018.10.011

Information Processing Letters 143 (2019) 4–7

Contents lists available at ScienceDirect

Information Processing Letters

www.elsevier.com/locate/ipl

Enhanced string factoring from alphabet orderings ☆,☆☆

Amanda Clare [a,*], Jacqueline W. Daykin [a,b,c]

[a] Department of Computer Science, Aberystwyth University, SY23 3DB, UK
[b] Department of Informatics, King's College London, WC2B 4BG, UK
[c] Department of Information Science, Stellenbosch University, South Africa

ARTICLE INFO

Article history:
Received 15 June 2018
Received in revised form 10 October 2018
Accepted 21 October 2018
Available online 22 October 2018
Communicated by Kun-Mao Chao

Keywords:
Alphabet ordering
Combinatorial problems
Design of algorithms
Greedy algorithm
Lyndon factorization

ABSTRACT

In this note we consider the concept of alphabet ordering in the context of string factoring. We propose a greedy algorithm that produces Lyndon factorizations with small numbers of factors which can be modified to produce large numbers of factors. For the technique we introduce the Exponent Parikh vector. Applications and research directions derived from circ-UMFFs are discussed.

1. Introduction

Factoring strings is a powerful form of the divide and conquer problem-solving paradigm for strings or words. Notably, the Lyndon factorization [1] is both efficient to compute [5] and useful in practice [7]. We study the effect of an alphabet's order on the number of factors in a Lyndon factorization and propose a greedy algorithm that assigns an ordering to the alphabet. In addition, we formalize the distinction between the sets of Lyndon and co-Lyndon words [3] as avenues for alternative string factorizations. More generally, circ-UMFFs provide the opportunity for achieving further diversity with string factors [2,3].

1.1. Notation

Given an integer $n \geq 1$ and a nonempty set of symbols Σ (bounded or unbounded), a ***string of length n***, equivalently ***word***, over Σ takes the form $\boldsymbol{x} = x_1 \ldots x_n$ with each $x_i \in \Sigma$. For brevity, we write $\boldsymbol{x} = \boldsymbol{x}[1..n]$ with $\boldsymbol{x}[i] = x_i$. The length n of a string $\boldsymbol{x}$ is denoted by $|\boldsymbol{x}|$. The set Σ is called an ***alphabet*** whose members are ***letters*** or ***characters***, and Σ^+ denotes the set of all nonempty finite strings over Σ. The ***empty string*** of length zero is denoted $\boldsymbol{\varepsilon}$; we write $\Sigma^* = \Sigma^+ \cup \{\boldsymbol{\varepsilon}\}$ and let $|\Sigma| = \sigma$. We use exponents to denote repetition, for instance if $\alpha \in \Sigma$ then α^3 means $\alpha\alpha\alpha$. If $\boldsymbol{x} = \boldsymbol{u}\boldsymbol{w}\boldsymbol{v}$ for strings $\boldsymbol{u}, \boldsymbol{w}, \boldsymbol{v} \in \Sigma^*$, then $\boldsymbol{u}$ is a ***prefix***, $\boldsymbol{w}$ is a ***substring*** or ***factor***, and $\boldsymbol{v}$ is a ***suffix*** of $\boldsymbol{x}$; we say $\boldsymbol{u} \neq \boldsymbol{x}$ is a ***proper prefix*** and similarly for the other terms. If $\boldsymbol{x} = \boldsymbol{u}\boldsymbol{v}$, then $\boldsymbol{v}\boldsymbol{u}$ is said to be a ***rotation*** (***cyclic shift*** or ***conjugate***) of $\boldsymbol{x}$. A string $\boldsymbol{x}$ is said to be a ***repetition*** if and only if it has a factorization $\boldsymbol{x} = \boldsymbol{u}^k$ for some integer $k > 1$; otherwise, $\boldsymbol{x}$ is said to be ***primitive***. For a string $\boldsymbol{x}$, the reversed string $\overline{\boldsymbol{x}}$ is defined as $\overline{\boldsymbol{x}} = \boldsymbol{x}[n]\boldsymbol{x}[n-1]\cdots\boldsymbol{x}[1]$. A string which is both a proper prefix and a proper suffix of a string $\boldsymbol{x} \neq \boldsymbol{\varepsilon}$ is called a ***border*** of $\boldsymbol{x}$; a string is ***border-free*** if the only border it has is the empty string $\boldsymbol{\varepsilon}$.

☆ The authors were part-funded by the European Regional Development Fund through the Welsh Government, grant 80761-AU-137 (West).
☆☆ A preliminary version of this paper was accepted as a poster in IWOCA 2018 (International Workshop on Combinatorial Algorithms).
* Corresponding author.
E-mail addresses: afc@aber.ac.uk (A. Clare), jwd6@aber.ac.uk, jackie.daykin@kcl.ac.uk (J.W. Daykin).

https://doi.org/10.1016/j.ipl.2018.10.011

If Σ is a totally ordered alphabet then ***lexicographic ordering*** (***lexorder***) $\boldsymbol{u} < \boldsymbol{v}$ with $\boldsymbol{u}, \boldsymbol{v} \in \Sigma^+$ means that either $\boldsymbol{u}$ is a proper prefix of $\boldsymbol{v}$, or $\boldsymbol{u} = \boldsymbol{ras}$, $\boldsymbol{v} = \boldsymbol{rbt}$ for some $a, b \in \Sigma$ such that $a < b$ and for some $\boldsymbol{r}, \boldsymbol{s}, \boldsymbol{t} \in \Sigma^*$. We call the ordering $\prec$ based on lexorder of reversed strings ***co-lexicographic ordering*** (***co-lexorder***). Using the Roman alphabet: $ca < cat < dog$ while $dog \prec cat$.

2. Unique Maximal Factorization Families (UMFFs)

A subset $\mathcal{W} \subseteq \Sigma^+$ is a ***factorization family*** (FF) if and only if for every nonempty string $\boldsymbol{x}$ on Σ there exists a factorization $F_{\mathcal{W}}(\boldsymbol{x})$ of $\boldsymbol{x}$ over $\mathcal{W}$. If every factor of $F_{\mathcal{W}}(\boldsymbol{x})$ is maximal (***max***) in length over $\mathcal{W}$ then the factorization is said to be max, and hence must be unique. So if $\mathcal{W}$ is an FF on an alphabet Σ then $\mathcal{W}$ is a ***unique maximal factorization family*** (UMFF) if there exists a max factorization $F_{\mathcal{W}}(\boldsymbol{x})$ for every string $\boldsymbol{x} \in \Sigma^+$ – for this theory see [2,3].

An UMFF $\mathcal{W}$ is a ***circ-UMFF*** if it contains exactly one rotation of every primitive string $\boldsymbol{x} \in \Sigma^+$. The classic and foundational circ-UMFF is the set of Lyndon words, which we denote $\mathcal{L}$, where the rotation chosen is the one that is strictly least in the lexorder derived from an ordering of the letters of the alphabet Σ ([1,5,2]). The co-Lyndon circ-UMFF, co-$\mathcal{L}$, was introduced in [3], where a co-Lyndon word is strictly least amongst its rotations in co-lexorder.

Every circ-UMFF $\mathcal{W}$ yields a strict order relation, the ***$\mathcal{W}$-order***: if $\mathcal{W}$ contains strings $\boldsymbol{u}$, $\boldsymbol{v}$ and $\boldsymbol{uv}$ then $\boldsymbol{u} <_{\mathcal{W}} \boldsymbol{v}$. For the Lyndon circ-UMFF, its specific $\mathcal{W}$-order is lexorder:

Theorem 1. *(Duval [5]) Suppose $\boldsymbol{u}, \boldsymbol{v} \in \mathcal{L}$, then $\boldsymbol{uv} \in \mathcal{L}$ if and only if $\boldsymbol{u} < \boldsymbol{v}$.*

It was observed in [3] that the analogue of Theorem 1 does not hold for every circ-UMFF – we will establish this phenomenon for the co-Lyndon circ-UMFF.

Lemma 1. *Suppose $\boldsymbol{u}, \boldsymbol{v} \in$ co-$\mathcal{L}$, then $\boldsymbol{uv} \in$ co-$\mathcal{L}$ if and only if $\boldsymbol{v} \prec \boldsymbol{u}$.*

Proof. Since $\boldsymbol{u}, \boldsymbol{v} \in$ co-$\mathcal{L}$ then $\overline{\boldsymbol{u}}, \overline{\boldsymbol{v}} \in \mathcal{L}$. If $\boldsymbol{v} \prec \boldsymbol{u}$ in co-lexorder then $\overline{\boldsymbol{v}} < \overline{\boldsymbol{u}}$ in lexorder. Applying Theorem 1 we have $\overline{\boldsymbol{vu}} \in \mathcal{L}$ and hence $\boldsymbol{uv} \in$ co-$\mathcal{L}$. Next if $\boldsymbol{uv} \in$ co-$\mathcal{L}$ then it must be primitive and border-free [2]. Thus $\boldsymbol{u} \neq \boldsymbol{v}$ which gives rise to two cases. Suppose first that $\boldsymbol{u} \prec \boldsymbol{v}$. If $\boldsymbol{u}$ is a proper suffix of $\boldsymbol{v}$ then $\boldsymbol{uv} = \boldsymbol{uwu}$ for some $\boldsymbol{w} \neq \varepsilon$ contradicting the border-free property. Otherwise, with $|\boldsymbol{u}| = n$ there is some largest j, $1 \leq j \leq n$, such that $\boldsymbol{u}[j] \neq \boldsymbol{v}[j]$. If $\boldsymbol{u}[j] < \boldsymbol{v}[j]$ then $\boldsymbol{vu} \prec \boldsymbol{uv}$ contradicting $\boldsymbol{uv} \in$ co-$\mathcal{L}$. We conclude that $\boldsymbol{u}[j] > \boldsymbol{v}[j]$, and so $\boldsymbol{v} \prec \boldsymbol{u}$ as required. □

The sets of Lyndon and co-Lyndon words are distinct and almost disjoint.

Lemma 2. *For a given Σ, $\mathcal{L} \neq$ co-$\mathcal{L}$ and $\mathcal{L} \cap$ co-$\mathcal{L} = \Sigma$.*

Proof. Let $\boldsymbol{v} \in \mathcal{L}$ and $\boldsymbol{w} \in$ co-$\mathcal{L}$ with $|\boldsymbol{v}|, |\boldsymbol{w}| \geq 2$. Then $\boldsymbol{v}$ starts with some letter α which is minimal in $\boldsymbol{v}$. Since $\boldsymbol{v}$ is border-free [2] then it ends with some β where $\alpha < \beta$. Similarly, $\boldsymbol{w}$ starts γ and ends δ, where $\gamma > \delta$. Therefore $\boldsymbol{v} \neq \boldsymbol{w}$. Finally, every circ-UMFF contains the alphabet Σ as expressed in [2,3]. □

The following result generalizes the Lyndon factorization theorem [1] and is a key to further applications of string decomposition.

Theorem 2. *[2] Let $\mathcal{W}$ be a circ-UMFF and suppose $\boldsymbol{x} = \boldsymbol{u_1 u_2 \cdots u_m}$, with each $\boldsymbol{u_j} \in \mathcal{W}$. Then $F_{\mathcal{W}}(\boldsymbol{x}) = \boldsymbol{u_1 u_2 \cdots u_m}$ iff $\boldsymbol{u_1} \geq_{\mathcal{W}} \boldsymbol{u_2} \geq_{\mathcal{W}} \ldots \geq_{\mathcal{W}} \boldsymbol{u_m}$.*

3. Alphabet ordering

Suppose the goal is to optimize a Lyndon factorization by minimizing or maximizing the number of factors. For this we consider choosing the order of the letters in the – assumed unordered – alphabet so as to influence the number of factors. To illustrate, consider the string $\boldsymbol{x} = abcabcdabcaba$. If $\Sigma = \{a < b < c < d\}$, then $F_{\mathcal{L}}(\boldsymbol{x}) = abcabcd \geq abc \geq ab \geq a$. Whereas, if we choose the alphabet ordering to be $\{b < c < a < d\}$, the Lyndon factorization of $\boldsymbol{x}$ becomes $a \geq bcabcdabcaba$.

Towards this goal we now describe a greedy algorithm for producing small numbers of factors which has performed well in practice on the biological $\{A, C, G, T\}$ alphabet – the experimentation compared results with those for the 4! letter permutations. For a string $\boldsymbol{v} = v_1 \ldots v_n$, we suppose that the number of distinct characters in $\boldsymbol{v}$ is $\delta \leq \sigma$; for practical purposes we can assume σ is at most $O(n)$.

The proposed method requires an extension to a Parikh vector, $p(\boldsymbol{v})$, of a finite word $\boldsymbol{v}$, where $p(\boldsymbol{v})$ enumerates the occurrences of each letter of the alphabet in $\boldsymbol{v}$. Our modification is that for each distinct letter we will record its individual RLE (run length encoding) exponent pattern (left-to-right sequence of exponents for a letter) – so the sum of these exponents is the Parikh entry for that letter. We call this the ***Exponent Parikh vector***, or EP vector, implemented as an EP array. For example, over the alphabet $\Sigma = \{b < c < d < f\}$, if $\boldsymbol{v} = bbbffbbcf$ then $p(\boldsymbol{v}) = [5, 1, 0, 3]$; whereas, for the EP vector we record the sequences $[(3, 2), (2, 1), (1)]$. So usually the letters are processed in the alphabet order with a Parikh vector while in the EP case we process them in order of first occurrence.

An overview of the method is that we use the fact that in a Lyndon factorization the first factor is the longest prefix which is a Lyndon word. Then the heuristic is that the left-most letter, α say, in the given string whose exponents, when read as a string, form a Lyndon word with the minimal number of factors is chosen as the least letter in the alphabet ordering. In order to construct a Lyndon word using the exponents of letters we require the order of the exponent integer alphabet to be inverted, that is, let $\bar{\Sigma} = \{\ldots 3 < 2 < 1\}$. Next, the algorithm attempts to assign order to letters in the substrings between runs of α characters, where these substrings are denoted X_i – if it gets stuck it tries backtracking. Finally, if there is a nonempty prefix prior to the first α then it is processed.

So note that with this algorithm the required property for the exponents of α is that they form a Lyndon

word over $\bar{\Sigma}$ and in conjunction a requirement for assigning letters to the X_i substrings is that the ordering will be cycle-free. The algorithm can be modified to generate large numbers of factors which involves assigning distinct letters to be in decreasing order.

3.1. Greedy algorithm

The pseudocode in Algorithm 1 greedily assigns an alphabet order to letters.

Algorithm 1: Order the alphabet so as to reduce the number of factors in a Lyndon factorization.

```
With a linear scan record the Exponent Parikh (EP) vector of the
string for δ distinct letters – O(n);
Compute F_L(p_r) of each exponent string p_r over Σ̄ and record its
number of factors – O(n);
bool ← true;                          // initiate alphabet ordering
while bool = true do
    Select the next leftmost p_r, p_i say, with minimal number of
    factors, t say – O(n);
    // assign alphabet order to the t factors of
       F_L(p_i) = f_1 ≥ ··· ≥ f_t
    // where f_j = α^{j_1} X_1 α^{j_2} X_2 ··· α^{j_q} X_q , and α ∉ X_h ,
       1 ≤ h ≤ q , with j_1 j_2 ... j_q ∈ L over Σ̄
    α = λ_1;    // assign first letter to be minimal in Σ; if q = 1 assign each new letter in X_1
    successively in Σ
    for h = 2 to q do
        if j_h = j_1 then     // same exponents so assign alphabet in order to letters in X_1 and X_h
        substrings
            d ← 1;
            while X_1[d] = X_h[d] do
                assign each new letter successively in Σ; d++;
            if X_1[d] = α & X_h[d] ≠ α then
                assign X_h[d] to be next successive letter;
            else if X_h[d] = α & X_1[d] ≠ α then
                bool ← false;                          // not Lyndon
            else if assignment would not make inconsistency then
            // X_1[d] ≠ X_h[d]
                assign X_h[d] > X_1[d];
            else
                bool ← false;                          // inconsistent
if bool then
    attempt assignment process for the t factors of F_L(p_i);
if bool then
    if string prefix u (prior to f_1) is non-empty then      // α ∉ u
        repeat process on u starting with next successive letters
        in Σ;
        // lookup EP vector
        complete assignment of any remaining letters;
        if letters in prefix u do not occur in suffix then re-assign
        all letters starting from prefix;
else
    arbitrarily choose next leftmost p_r with minimal number of
    factors and attempt new assignment;
```

The following example, which uses the notation of Algorithm 1, illustrates how backtracking can lead the algorithm from an inconsistent ordering to a successful assignment and associated factorization.

Example 1. Assume $\Sigma = \{a, b, c, d\}$ and

$$\boldsymbol{x} = a^2bdca^2cda^2bdba^1ba^2bca^2ca^2ca^1b.$$

Only the letter a has an exponent greater than 1, and $F_{\mathcal{L}}(EP(a)) = 2221 \geq 2221$ has the minimal number of factors with $\boldsymbol{f_1} = \boldsymbol{f_2} = 2221$. After assigning a to be the first letter in Σ, processing $\boldsymbol{f_1}$ causes inconsistency (and similarly $\boldsymbol{f_2}$), since the substrings X_1 and X_2 give $b < c$ while X_1 and X_3 would require $c < b$. So the algorithm then backtracks through the EP array and chooses the letter with the least number of factors (albeit singletons), namely d – the result is $\Sigma = \{d < c < a < b\}$ with $F_{\mathcal{L}}(\boldsymbol{x}) = aab \geq dcaacdaabdbabaabcaacaacab$. Note the order $\{a < b < c < d\}$ would have given 3 factors.

4. Experimentation: factorization of DNA strings

We chose as an example the 120 prokaryotic reference genomes from RefSeq,[1] to investigate the results of the algorithm in practice.[2] Most of these genomes are provided as a single contiguous sequence but some of them have additional smaller pieces representing plasmids or other information. The longest contiguous sequence was chosen for each genome in these cases, and smaller pieces were discarded. The retained sequences ranged from 640,681 letters to 10,236,715 in length, with a mean of 3,629,792.

In order to determine how often our greedy algorithm found a good or optimal alphabet reordering in practice, we calculated the Lyndon factorizations resulting from all possible $(4! = 24)$ alphabet reorderings of the characters A, C, G and T across this collection of genomes. The improvement that could potentially be made to the factorization by reordering is substantial, with at least a halving of the number of factors in most cases and an improvement reducing 25 factors down to 3 in one case. For each genome we ranked the results of all possible reorderings by the number of factors produced and compared the reordering produced by the algorithm. The algorithm found the optimal reordering for 21/120 genomes and the second-most optimal in 31/120 genomes.

The EP vector is used to determine the least letter in the reordering. If the first choice leads to inconsistency (and hence small factors), backtracking to inspect other possible choices can be helpful. However, in many cases, the initial choice is still better than the next possible consistent solution found via backtracking through the EP vector. Without backtracking, the algorithm found 23/120 optimal orderings and a further second-most optimal orderings in 31/120 genomes. Histograms of the full results, with and without backtracking, can be seen in the Supplemental Information.

5. Applications

In many cases, such as natural language text processing, the order of the alphabet is prescribed, and hence

[1] RefSeq: https://www.ncbi.nlm.nih.gov/refseq/about/prokaryotes.
[2] Code available at https://github.com/amandaclare/lyndon-factors.

the Lyndon factors of an input text cannot be manipulated. On the other hand, bioinformatics alphabets have no inherent ordering suggested by biological systems and applications involving Lyndon words, such as the Burrows–Wheeler transform (BWT), will allow for useful manipulation of the Lyndon factors. The co-BWT is the regular BWT of the reversed string, or the BWT with co-lexorder, which has been applied in the highly successful Bowtie sequence alignment program [6]. Integral with the BWT transform is the computation of suffix arrays via induced suffix-sorting. We also propose that pattern matching can be implemented with the Lyndon factorization in big data applications, such as sequence alignment, and further enhanced by fortuitous arrangements of the alphabet.

In [7] a new method is presented for constructing the suffix array of a text by using its Lyndon factorization advantageously. Partitioning the text according to its Lyndon properties allows tackling the problem in local portions of the text, local suffixes, prior to extending the solution globally, to achieve the suffix array of the entire text. It is stated that the algorithm's time complexity is not competitive for the construction of the overall suffix array – we propose that manipulating the factors by alphabet ordering could improve the efficiency.

6. Research problems

- As a complementary structure to the Lyndon array we introduce and propose studies of the Lyndon factorization array. The ***Lyndon array*** $\lambda = \lambda_{\boldsymbol{x}}[1..n]$ of a given $\boldsymbol{x} = \boldsymbol{x}[1..n]$ gives at each i the length of the maximal Lyndon word starting at i – reverse engineering in [4] includes a linear-time test for whether an integer array is a Lyndon array. We define the ***Lyndon factorization array*** $\boldsymbol{F} = \boldsymbol{F}_{\boldsymbol{x}}[1..n]$ of $\boldsymbol{x}$ to give at each position i the number of factors in the Lyndon factorization starting at i.
- The greedy algorithm does not necessarily produce an optimal solution hence natural problems are to design algorithms for Lyndon factorizations with a guaranteed minimal/maximal number of Lyndon factors.
- Using Duval's Lyndon factorization algorithm [5] as a benchmark, modify the alphabet order so as to increase/decrease the number of factors.
- Theorem 2 supports the following optimization problem from [3]: Determine the circ-UMFF(s) which factors a string $\boldsymbol{x}$ into the minimal/maximal number of factors, possibly combined with alphabet ordering.

Appendix A. Supplementary material

Supplementary material related to this article can be found online at https://doi.org/10.1016/j.ipl.2018.10.011.

References

[1] K.T. Chen, R.H. Fox, R.C. Lyndon, Free differential calculus, IV. The quotient groups of the lower central series, Ann. Math. 68 (1) (1958) 81–95.

[2] D.E. Daykin, J.W. Daykin, Properties and construction of unique maximal factorization families for strings, Int. J. Found. Comput. Sci. 19 (4) (2008) 1073–1084.

[3] D.E. Daykin, J.W. Daykin, W.F. Smyth, Combinatorics of unique maximal factorization families (UMFFs), in: Special Issue on Stringology, Fundam. Inform. 97 (3) (2009) 295–309.

[4] J.W. Daykin, F. Franek, J. Holub, A.S.M.S. Islam, W.F. Smyth, Reconstructing a string from its Lyndon arrays, Theor. Comput. Sci. 710 (2018) 44–51.

[5] J.-P. Duval, Factorizing words over an ordered alphabet, J. Algorithms 4 (4) (1983) 363–381.

[6] B. Langmead, C. Trapnell, M. Pop, S.L. Salzberg, Ultrafast and memory-efficient alignment of short DNA sequences to the human genome, Genome Biol. 10 (3) (2009) R25.

[7] S. Mantaci, A. Restivo, G. Rosone, M. Sciortino, Suffix array and Lyndon factorization of a text, J. Discret. Algorithms 28 (2014) 2–8.