On Baier's Sort of Maximal Lyndon Substrings

Frantisek Franek[1], Michael Liut[1], and W. F. Smyth[1,2]

[1] Department of Computing and Software
McMaster University, Hamilton, Canada
{franek/liutm/smyth}@mcmaster.ca
[2] School of Engineering and Information Technology
Murdoch University, Perth, Australia

Abstract. We describe and analyze in terms of Lyndon words an elementary sort of maximal Lyndon factors of a string and prove formally its correctness. Since the sort is based on the first phase of Baier's algorithm for sorting of the suffixes of a string, we refer to it as Baier's sort.

Keywords: string, suffix, suffix array, Lyndon array, Lyndon string, maximal Lyndon substring

1 Introduction

The computation of the maximal Lyndon substrings of a string has been actively researched since Bannai et al. presented a linear-time algorithm for computing runs [3]. Their algorithm relies on knowledge of all maximal Lyndon factors with respect to an order of the alphabet, and knowledge of all maximal Lyndon factors with respect to the inverse order. The maximal Lyndon factors of a string $x = x[1\,..\,n]$ are represented in the *Lyndon array* $L[1\,..\,n]$, where $L[i] =$ *the length of the maximal Lyndon factor starting at position* i: see [7,11] and references therein. Other linear-time algorithms for computing all the runs rely on Lempel-Ziv factorization, and so the comparative efficiency of the runs algorithms is determined by the comparison of computing the Lyndon array and computing the Lempel-Ziv factorization: see [6,4] and references therein.

There is only one known linear-time algorithm for computing the Lyndon array and it relies on suffix sorting: compute the suffix array, then the inverse suffix array, and then employ NSV (Next Smaller Value) algorithm to compute the Lyndon array from the inverse suffix array [12]. All other algorithms have $\mathcal{O}(n \log n)$ or $\mathcal{O}(n^2)$ worst case complexity: see [9,14] and references therein. The Lempel-Ziv factorization can be efficiently computed in linear time. Thus, our research focuses on linear computation of the Lyndon array without the need to build an unrelated global data structure such as the suffix array.

Baier [1,2] in 2016 presented an elementary though elaborate algorithm for suffix sorting.The algorithm works in two phases. Soon afterwards, Cristoph Diegelmann in

Frantisek Franek, Michael Liut, W. F. Smyth: On Baier's Sort of Maximal Lyndon Substrings, pp. 63–78.
Proceedings of PSC 2018, Jan Holub and Jan Žďárek (Eds.), ISBN 978-80-01-06484-9

reaction to [9] was first to note that phase I of the Baier's suffix sorting algorithm in fact identifies and sorts the maximal Lyndon factors in linear time [10]. The performance of the algorithm was not great: Baier himself noted that his implementation of the suffix sort was about four times slower than the best linear algorithms for suffix sorting. Our analysis indicated that the first phase was the main culprit. We analyzed phase II in detail and presented a different implementation in [10].

There are three main goals for our study of Baier's sort. The first is to describe and formalize Baier's sort in terms of Lyndon words. The second is to provide a more formal and more detailed proof of the correctness of the sort in the framework of Lyndon words. And finally, the third goal is a detailed analysis of the method essential for a more efficient programming implementation in order to speed up the execution and lower the memory required. An additional goal is to see whether knowledge of the Lyndon array for a string can be utilized to speed up Baier's sort of the maximal Lyndon factors. This paper concerns the first two goals.

Lyndon words admit the standard factorization and so often are built from these two components in what is in essence a bottom-up approach [3,16]. However, maximal Lyndon factors of a string can be built in a top-down fashion not related to the standard factorization. In Baier's sort, the maximal Lyndon factors are determined using what we call the *water draining method*[1] since the process can be best visualized as if the string represented a bunch of hills in a water tank and we slowly drain the water from the tank. The maximal Lyndon factors are then built from the hills that the draining reveals. The *water draining method* has three steps:

(a) lower the water level by one
(b) extend the existing Lyndon factors
the revealed letters are used to extend the existing Lyndon factors where possible, or became Lyndon factors of length 1 otherwise;
(c) consolidate the new Lyndon factors
processed from the right, if several Lyndon factors are adjacent and can be joined to a longer Lyndon factor, they are joined.

In Fig. 1, we illustrate the process:

(1) We start with the string *abcdedbcdba* and a full tank of water.
(2) We drain one level, only e is revealed, nothing to extend, nothing to consolidate.
(3) We drain one more level and three d's are revealed, the first d extends e to de and the remaining two d's form Lyndon factors d of length 1, nothing to consolidate.
(4) We drain one more level and two c's are revealed, the first extends de to cde and the second extends d to cd, the consolidation then joins cde and d to $cded$ (5).

[1] This is not a pun on "graindraining" string used by Baier for illustration of his method.

(6) We drain one more level and three b's are revealed, the first extends $cded$ to $bcded$, the second extends cd to bcd, the third is a Lyndon factor b of length 1, nothing to consolidate.

(7) We drain one more level and two a's are revealed, the first extends $bcded$ to $abcded$ and the second becomes a Lyndon factor a of length 1, in (8) $abcded$ and bcd are joined to $abcdedbcd$, and we continue the consolidation in (9) where $abcdedbcd$ and b are joined to $abcdedbcdb$, and the consolidation is complete.

So, during the process the following maximal Lyndon factors were identified: e at position 5, de at position 4, d at positions 6, 9, $cded$ at position 3, cd at position 8, $bcded$ at position 2, bcd at position 7, b at position 10, $abcdedbcdb$ at position 1, and a at position 11. Note that all positions are accounted for, we really got all maximal Lyndon factors of the string $abcdedbcdba$. For a depiction of all maximal Lyndon factors of $abcdedbcdba$, see Fig. 2.

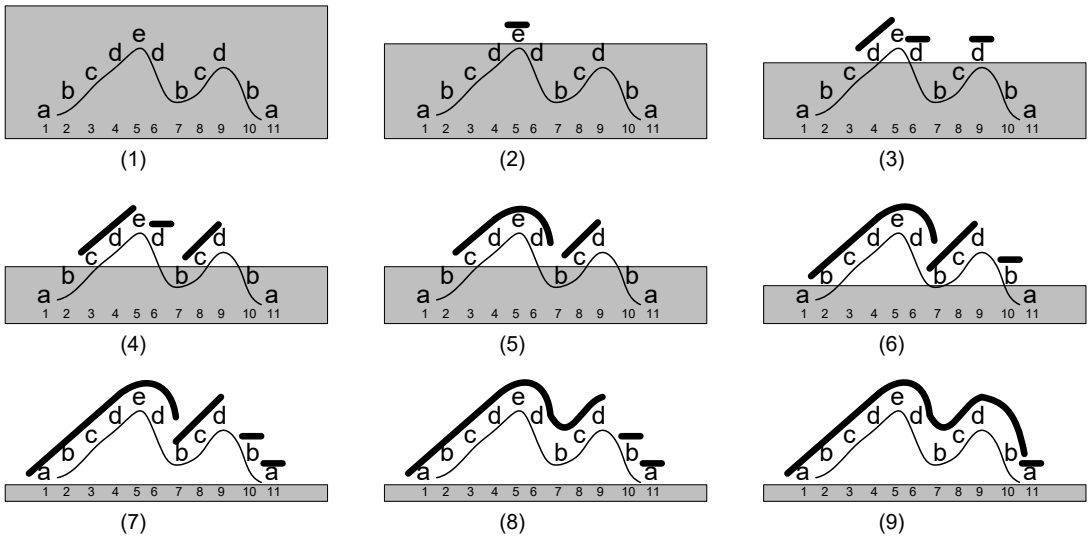


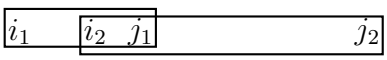
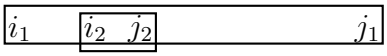
Figure 1. The water draining method for $abcdedbcdba$

The true ingenuity of Baier was to realize the water draining method by a simple linear mechanism based on the $prev()$ operator.

2 Background notation, notions and facts

In this section, we provide the description and definitions of the basic string notions and notation we are using in this paper, and the relevant basic facts. A ***string*** x is a sequence x[1..n] of ***letters*** x[i], 1 ¡ i ¡ n, each drawn from a set $\mathcal{A}$ called the ***alphabet***. The ***length*** of the string is n. $\mathcal{A}$ is a totally ordered set.

- The symbol ε denotes the empty string.
- A concatenation of two strings x and y is denoted by xy. A concatenation of k copies of u is is denoted as u^k, for an integer $k \geq 2$.
- If a string $x = uvw$, then u is a ***prefix*** of x, v is a ***factor*** or ***substring*** of x, and w is a ***suffix*** of x. A prefix (resp. suffix) is ***trivial*** if it is empty, and is ***proper*** if it is not the whole x.
- The fact that u is a prefix of x is denoted by $u \trianglelefteq x$, while the fact that u is a proper prefix of x is denoted by $u \triangleleft x$.
- A string w is a ***border*** of a string x, if w is both a proper prefix and a proper suffix of x. An empty string is a ***primitive*** border. If x has only a primitive border, it is ***unbordered***.
- A string x is ***primitive*** if there are no integer $k \geq 2$ and string u so that $x = u^k$.
- The expression $x \prec y$ denotes the fact that x is *lexicographically* smaller than y; that is, either x is a proper prefix of y or there is $i \leq \min\{|x|, |y|\}$ such that $x[1\,..\,i{-}1] = y[1\,..\,i{-}1]$ and $x[i] \prec y[i]$.
- The expression $x \preccurlyeq y$ denotes the fact that $x \prec y$ or $x = y$.
- The expression $x \prec\!\!\!\triangleleft\, y$ denotes the fact that $x \prec y$, but x is not a prefix of y.
- For a string $x = wu$, uw is called a ***rotation*** of x, if either u or w is empty, than the rotation is ***trivial***.
- A string x is ***Lyndon*** if either $|x| = 1$, the so-called ***trivial*** Lyndon string, or x is lexicographically strictly smaller than any of its non-trivial rotations. Such a string is often called a *Lyndon word* [5].
- A Lyndon factor $u = x[i\,..\,j]$ of a string $x = x[1\,..\,n]$ is ***maximal*** if either $j = n$ or for any $j < \ell \leq n$, $x[i\,..\,\ell]$ is not Lyndon.
- For a string x, $\mathcal{A}_x$ denotes the ***alphabet*** of the string; that is, the set of letters occurring in x.
- For two factors $x[i_1\,..\,j_1]$ and $x[i_2\,..\,j_2]$ a string $x = x[1\,..\,n]$ with $i_1 \leq i_2$, we say that these two factors

  - are ***disjoint***, if $j_1 < i_2$ 
  - ***overlap***, if $i_2 \leq j_1$
  - ***intersect***, if $i_2 \leq j_1 \leq j_2$ 
  - the first ***includes*** the second (or, the second is ***included*** in the first), if $j_2 \leq j_1$ 

  Note that in our terminology

  $$\big(x \text{ and } y \textit{ overlap}\big) \Leftrightarrow \big((x \text{ and } y \textit{ intersect}) \text{ or } (\text{one includes the other})\big)$$

- A family of factors of a string x has the ***Monge*** property, if each two distinct factors from the family are either disjoint or one includes the other, or, equivalently, no two factors intersect.

Facts 1. Basic facts of Lyndon strings ([8,12,13,15])

(i) *For u of length > 1,*
- *u is Lyndon $\Rightarrow$ u is unbordered $\Rightarrow$ u is primitive*
- *u is Lyndon iff $u \prec u_2$ for any $u = u_1u_2$*
- *u is Lyndon iff $u_1 \prec u_2$ for any $u = u_1u_2$*
- *u is Lyndon $\Rightarrow$ there are Lyndon words u_1, u_2 so that $u_1 \prec u_2$ and $u = u_1u_2$. If u_2 is the largest possible such suffix, it is called the **standard factorization** of u.*

(ii) *Let $u = wcv$ be Lyndon and let $c \prec d$. Then $uwd = wcvwd$ is Lyndon.*

(iii) *Let $x = vu$ for non-empty v and u and let v be a Lyndon factor of x. Then v is a maximal Lyndon factor of x iff $u \prec vu$.*

(iv) *Let $u = u_1vu_2$ be Lyndon and let v be a maximal Lyndon factor of u. Then $u \prec v$.*

3 Basic Definitions and Notions

In this section we present the basic definitions and notions for the description and analysis of Baier's sort.

Definition 2. *Let $x = x[1\,..\,n]$ be a string.*

- *A **group** with a **context** u, denoted as G_u, is an ascending sequence of indices $i_1 < \cdots < i_k$ such that u is a prefix of $x[i_\ell\,..\,n]$ for every $\ell \in 1\,..\,k$.*
- *A sequence $\mathscr{C} = [G_{u(m)}, G_{u(m-1)}, \ldots, G_{u(1)}]$ is a **group configuration** for x if*
  (i) *for any $\ell \in 1\,..\,m$, $G_{u(\ell)}$ is a group with the context $u(\ell)$, and*
  (ii) *$u(m) \prec u(m-1) \prec \cdots \prec u(1)$, and*
  (iii) *for any $\ell \in 1\,..\,m$, $u(\ell)$ is a Lyndon substring of x, and*
  (iv) *all the groups are pairwise mutually disjoint, and*
  (v) *$\bigcup_{\ell=1}^{m} G_{u(\ell)} = 1\,..\,n$, and*
  (vi) *the family $\{\ \mathscr{C}\langle i\rangle \mid 1 \le i \le n\ \}$ has the Monge property, where $\mathscr{C}\langle i\rangle = x[i\,..\,i+|u(\ell)|-1]$ for the unique ℓ such that $i \in G_{u(\ell)}$, note that $\mathscr{C}\langle i\rangle = u(\ell)$.*
- *For $i, j \in G_{u(\ell)}$, $i \otimes j$ iff $|i - j| = |u(\ell)|$. The relation $\sim$ is defined as a transitive closure of $\otimes$. Thus, $i \sim j$ is an equivalence relation on $G_{u(\ell)}$. The symbol $[i]_\sim$ denotes the class of equivalence $\sim$ to which i belongs.*
- *For $i \in G_{u(\ell)}$, the **valence** of i is defined as $val_{\mathscr{C}}(i) = |[i]_\sim|$.*
- *$gr_{\mathscr{C}}(i)$ denotes the unique group of $\mathscr{C}$ the index i belongs to, i.e. $gr_{\mathscr{C}}(i) = G_{u(\ell)}$ iff $i \in G_{u(\ell)}$.*
- *A group $G_{u(\ell)}$ is **complete** if*
  - for any $i \in G_{u(\ell)}$, $\mathscr{C}\langle i\rangle$ is a maximal Lyndon factor of x, and
  - if $u(\ell) = x[j\,..\,j+|u|-1]$ is a maximal Lyndon factor of x, then $j \in G_{u(\ell)}$.

- *The operator* $prev_{\mathscr{C}}(i) = \max\{j < i \mid con_{\mathscr{C}}(gr_{\mathscr{C}}(j)) \prec con_{\mathscr{C}}(gr_{\mathscr{C}}(i))\}$*, if such a* j *exists,* `nil` *otherwise, where* $con_{\mathscr{C}}(G)$ *denotes the context of group* G *with respect to the configuration* $\mathscr{C}$*.*

Note that it is possible for an $i \in G_u$, that not only $x[i\,..\,i+|u|-1] = u$, but also $x[i+|u|\,..\,x[i+2|u|-1] = u$ and so forth. This is captured by the notion of the valence: if $i \in G_u$ and $val(i) = k$, the occurrence of u at position i is a part of a maximal repetition u^k.

It is easy to see, that for $j \in [i]_{\sim}$, $prev_{\mathscr{C}}(i) = prev_{\mathscr{C}}(j)$. Let $i, j \in G_u$. WLOG assume that $j = i+p|u|$ and so $x[i\,..\,j+|u|-1] = u^{p+1}$. Clearly, $prev_{\mathscr{C}}(i) \leq prev_{\mathscr{C}}(j)$. If $prev_{\mathscr{C}}(j) > prev_{\mathscr{C}}(i)$, it would indicate that a suffix of u must be lexicographically smaller than u, which contradicts the Lyndon property of u.

Together, (iv) and (v) assure that the groups of a group configuration form a disjoint partitioning of the string's index range $1\,..\,n$.

Since Definition 2 is quite involved, we present and illustrative example on a string $x = x[1..11] = abcdedbcdba$.

1	2	3	4	5	6	7	8	9	10	11
a	*b*	*c*	*d*	*e*	*d*	*b*	*c*	*d*	*b*	*a*

$G_a = \{11\}$, $G_{abcdedbcdb} = \{1\}$, $G_b = \{10\}$, $G_{bcd} = \{7\}$, $G_{bcded} = \{2\}$,
$G_{cd} = \{8\}$, $G_{cded} = \{3\}$, $G_d = \{6, 9\}$, $G_{de} = \{4\}$, $G_e = \{5\}$.

Take for instance the group $G_d = \{6, 9\}$, the context of the group is a string of length 1, d, and indeed, at the positions 6 and 9 we have substrings d starting. Moreover, G_d is complete, as $x[6]$ and $x[9]$ are both maximal Lyndon, and there is no other occurrence of a maximal Lyndon substring d. Similarly for $G_{abcdedbcdb} = \{1\}$, $x[1..10] = abcdedbcdb$, so it is the context, $abcdedbcdb$ is Lyndon, and $x[1..10] = abcdedbcdb$ is maximal, and there is no other occurrence of maximal $abcdedbcdb$, so $G_{abcdedbcdb}$ is complete. It is easy to see, that the whole set of the groups in the order given above is a group configuration: $a \prec abcdedbcdb \prec b \prec bcd \prec bcded \prec cd \prec cded \prec d \prec de \prec e$ and $G_a \cup G_{abcdedbcdb} \cup G_b \cup G_{bcd} \cup G_{bcded} \cup G_{cd} \cup G_{cded} \cup G_d \cup G_{de} \cup G_e = 1\,..\,11$.

Denote $\mathscr{C} = [G_a, G_{abcdedbcdb}, G_b, G_{bcd}, G_{bcded}, G_{cd}, G_{cded}, G_d, G_{de}, G_e]$. Consider the system of factors $\{\ \mathscr{C}\langle i\rangle \mid 1 \leq i \leq n\ \}$, let us demonstrate that it indeed has the Monge property:

$\mathscr{C}\langle 1\rangle = x[1\,..\,10] = abcdedbcdb$, $\mathscr{C}\langle 2\rangle = x[2\,..\,6] = bcded$,
$\mathscr{C}\langle 3\rangle = x[3\,..\,6] = cded$, $\mathscr{C}\langle 4\rangle = x[4\,..\,5] = de$,
$\mathscr{C}\langle 5\rangle = x[5] = e$, $\mathscr{C}\langle 6\rangle = x[6] = d$, $\mathscr{C}\langle 7\rangle = x[7\,..\,9] = bcd$,

$\mathscr{C}\langle 8\rangle = x[8\,..\,9] = cd$, $\mathscr{C}\langle 9\rangle = x[9] = d$, $\mathscr{C}\langle 10\rangle = x[10] = b$,
$\mathscr{C}\langle 11\rangle = x[11] = a$.

For instance, $\mathscr{C}\langle 1\rangle = x[1\,..\,10]$ includes $\mathscr{C}\langle 10\rangle = x[10]$, or $\mathscr{C}\langle 9\rangle = x[9]$ and $\mathscr{C}\langle 10\rangle = x[10]$ are disjoint, etc.

Lemma 3. *Let G_u be a group with a primitive context u for a string x. Let $i \in G_u$. Then $[i]_\sim = \{\ell \in G_u \mid \min [i]_\sim \le \ell \le \max [i]_\sim\}$.*

Proof. It suffices to prove that if $j - i = |u|$, then there is no $\ell \in G_u$ so that $i < \ell < j$. Assume there is such an ℓ. Then $x[i\,..\,i{+}|u|{-}1] = x[j\,..\,j{+}|u|{-}1] = u$ and since $j = i{+}|u|$, we have $x[i\,..\,i{+}2|u|{-}1] = 2u$. Moreover, $x[\ell\,..\,\ell{+}|u|{-}1] = u$ and $i < \ell < 2i$. Since u is primitive, this contradicts the synchronization principle for primitive strings. □

To illustrate Lemma 3, consider *abbbbabb*: $G_b = \{2, 3, 4, 5, 7, 8\}$. Consider $[3]_\sim = \{2, 3, 4, 5\}$, you can see that $[3]_\sim = \{i \in G_b \mid 2 \le i \le 5\}$. Similarly, $[7]_\sim = \{7, 8\} = \{i \in G_b \mid 7 \le i \le 8\}$.

Definition 4. *A group configuration $\mathscr{C} = [G_{u(m)}, G_{u(m-1)}, \ldots, G_{u(1)}]$ for a string x is r-**proper** for $1 \le r \le m$ if*

(vii) *all the groups $G_{u(r)}, \ldots, G_{u(1)}$ are complete; and*
(viii) *all the groups $G_{u(r-1)}, \ldots, G_{u(1)}$ have been processed; and*
(ix) *for any $i \in G_{u(r)}$, if $j = prev_{\mathscr{C}}(i)$, then $\mathscr{C}\langle j\rangle(\mathscr{C}\langle i\rangle)^t$ where $t = val_{\mathscr{C}}(i)$, is a prefix of $x[j\,..\,n]$.*

*To **process** the group $G_{u(r)}$ entails:*

(a) *Compute $P = \{prev_{\mathscr{C}_r}(i) \mid i \in G_{u(r)}$ and $prev_{\mathscr{C}_r}(i) \neq \mathtt{nil}\}$;*
(b) *Let $\approx$ be an equivalence on P defined by $i_1 \approx i_2$ iff $gr_{\mathscr{C}_r}(i_1) = gr_{\mathscr{C}_r}(i_2)$. Compute the disjoint partitioning of $P = P_1 \cup P_1 \cup \cdots \cup P_k$ into the classes of equivalence $\approx$. Let $P_1 \subseteq G_{u(n_1)}$, …, $P_k \subseteq G_{u(n_k)}$;*
(c) *For each $j \in 1\,..\,k$, let $\approx_j$ be an equivalence on P_j defined by $i_1 \approx_j i_2$ iff $i_1 = prev_{\mathscr{C}}(\ell_1)$ & $i_2 = prev_{\mathscr{C}}(\ell_2)$ & $val(\ell_1) = val(\ell_2)$. Let $[i_1]_{\approx_j}$ denote the class of equivalence of $\approx_j$ containing i_1. Define $val_{\mathscr{C}_r}([i_1]_{\approx_j}) = val(\ell_1)$ so that $i_1 = prev_{\mathscr{C}}(\ell_1)$. Compute the disjoint partitioning $P_j = P_{j,1} \cup P_{j,2} \cup \cdots \cup P_{j,t_j}$ into the classes of equivalence $\approx_j$. Moreover, let $val(P_{j,1}) > val(P_{j,2}) > \cdots > val(P_{j,t_j})$.*
(d) *for each $j \in 1\,..\,k$, for each $\ell \in 1\,..\,t_j$, move all indices of $P_{j,\ell}$ from $G_{u(n_j)}$ to a new group G'. The group G' is placed directly after $G_{u(n_j)}$ which is removed if it becomes empty. The context of G' is set to $u_{n_j}(u_r)^{val(P_{j,\ell})}$.*

Remark: Definition 4 may not seem sound, for the definition of *process* is only provided after it had been used in (viii). However, the definition is in fact recursive; that is for a 1-proper configuration, no group needs to be processed, then we process the last group to obtain a 2-proper configuration, and so on.

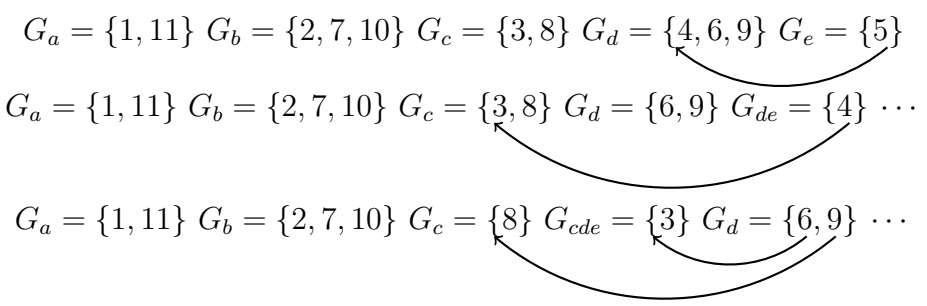
We will use a simple string *aabbabb* to illustrate the notions of the equivalence $\sim$, and the valence:

$$\begin{array}{c} 1\ 2\ 3\ 4\ 5\ 6\ 7 \\ a\ a\ b\ b\ a\ b\ b \end{array}$$

First consider the configuration $\mathscr{C}_1 = [G_a, G_b]$ where $G_a = \{1, 2, 5\}$ and $G_b = \{3, 4, 6, 7\}$. Then $val_{\mathscr{C}_1}(1) = val_{\mathscr{C}_1}(2) = val_{\mathscr{C}_1}(3) = val_{\mathscr{C}_1}(4) = val_{\mathscr{C}_1}(6) = val_{\mathscr{C}_1}(7) = 2$ and $val_{\mathscr{C}_1}(5) = 1$. It means, that for the processing of G_b we do not consider 4 and 7, only 3 and 6 and their valences of 2. Thus $P = \{2, 5\}$ as $2 = prev_{\mathscr{C}_1}(3) = prev_{\mathscr{C}_1}(4)$ and $5 = prev_{\mathscr{C}_1}(6) = prev_{\mathscr{C}_1}(7)$. Since $gr_{\mathscr{C}_1}(2) = gr_{\mathscr{C}_1}(5) = G_a$, 2 and 5 are both in the same P_j. So, after the refinement of G_a we get a new configuration $\mathscr{C}_2 = [G_a, G_{abb}, G_b]$ where $G_a = \{1\}$, $G_{abb} = \{2, 5\}$, and $G_b = \{3, 4, 6, 7\}$. Then $val_{\mathscr{C}_2}(2) = val_{\mathscr{C}_2}(5) = 2$, so $P = \{1\}$, as $1 = prev_{\mathscr{C}_2}(2) = prev_{\mathscr{C}_2}(5)$, and so we get a new and final configuration $\mathscr{C}_3 = [G_{aabbabb}, G_{abb}, G_b]$ where $G_{aabbabb} = \{1\}$, $G_{abb} = \{2, 5\}$, and $G_b = \{3, 4, 6, 7\}$.

For illustration, let us perform complete Baier's sort of *abcdedbcdba*, the arrows represent the *prev* operator. We start with the initial group configuration where each group groups all indices that start with the same letter. Processing from right, take the first unprocessed group from right, compute the *prev* values for indices in that group, and perform context concatenation wherever it points, partitioning the groups in the process. Then move to the next unprocessed group, until all groups except the very first one are processed.

$$\begin{array}{c} 1\ 2\ 3\ 4\ 5\ 6\ 7\ 8\ 9\ 10\ 11 \\ a\ b\ c\ d\ e\ d\ b\ c\ d\ b\ a \end{array}$$



$$G_a = \{1, 11\}\ G_b = \{2, 7, 10\}\ G_{cd} = \{8\}\ G_{cded} = \{3\}\ \cdots$$

$$G_a = \{1, 11\}\ G_b = \{7, 10\}\ G_{bcded} = \{2\}\ G_{cd} = \{8\}\ \cdots$$

$$G_a = \{1, 11\}\ G_b = \{10\}\ G_{bcd} = \{7\}\ G_{bcded} = \{2\}\ \cdots$$

$$G_a = \{11\}\ G_{abcded} = \{1\}\ G_b = \{10\}\ G_{bcd} = \{7\}\ \cdots$$

$$G_a = \{11\}\ G_{abcded} = \{1\}\ G_b = \{10\}\ \cdots$$

$$G_a = \{11\}\ G_{abcdedb} = \{1\}\ \cdots$$

$$G_a = \{11\}\ \cdots$$

The processed (in bold) classes represent not only all maximal Lyndon factors of the string, but they are also lexicographically ordered:

$$G_a = \{11\}, G_{abcdedb} = \{1\}, G_b = \{10\}, G_{bcd} = \{7\}, G_{bcdded} = \{2\},$$
$$G_{cd} = \{8\}, G_{cded} = \{3\}, G_d = \{6, 9\}, G_{de} = \{4\}, G_e = \{5\}.$$

Compare it to Fig. 2 to see that all maximal Lyndon factors are accounted for.

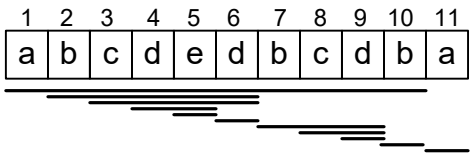


Figure 2. Maximal Lyndon Factors of *abcdedbcdba*

4 Properties of the group refinement

In this section we deal with the basic arrangement of group configurations referred in the text as ($\mathcal{B}$) given below:

Given a string $x[1\,..\,n]$ with the alphabet $\{a_1, \ldots, a_k\}$. For $\hat{r} \geq 1$,

$$\begin{aligned} &\mathscr{C}_1 = \left[Gu_{(1,m_1)}, Gu_{(1,m_1-1)} \ldots, Gu_{(1,1)}\right] \\ &\ldots \\ &\mathscr{C}_{\hat{r}} = \left[Gu_{(\hat{r},m_{\hat{r}})}, Gu_{(\hat{r},m_{\hat{r}}-1)} \ldots, Gu_{(\hat{r},1)}\right] \end{aligned} \qquad (\mathcal{B})$$

where $\mathscr{C}_1 = \left[Gu_{(1,m_1)}, Gu_{(1,m_1-1)} \ldots, Gu_{(1,1)}\right] = [G_{a_1}, \ldots, G_{a_k}]$;
for each $1 \leq r < \hat{r}$, $\mathscr{C}_r$ is an r-proper group configuration; and the

configuration $\mathscr{C}_{r+1} = \big[G_{u_{(r+1,\, m_{r+1})}}, G_{u_{(r+1,\, m_{r+1}-1)}} \ldots, G_{u_{(r+1,\, 1)}}\big]$ is produced by processing of the group $G_{u_{(r,\, r)}}$.

First, a few fundamental observations of the nature of "refinement" of the groups during processing.

Observation 5. *Referring to* $(\mathcal{B})$*, for any* $2 \le r \le \hat{r}$ *and any* $1 \le i, j \le n$*,*

(i) *If* $\mathscr{C}_r\langle i\rangle$ *is a proper suffix of* $\mathscr{C}_r\langle j\rangle$*, then* $\mathscr{C}_r\langle i\rangle = u_{(r,\, \xi)}$ *and* $\mathscr{C}_r\langle j\rangle = \mathscr{C}_\xi\langle j\rangle (u_{(r,\, \xi)})^t$ *where* $\xi < r$ *and* $t = val_{\mathscr{C}_\xi}(i)$*.*

(ii) *Either* $\mathscr{C}_r\langle i\rangle = \mathscr{C}_{r-1}\langle i\rangle$ *or* $\mathscr{C}_r\langle i\rangle = \mathscr{C}_{r-1}\langle i\rangle \big(u_{(r-1,\, r-1)}\big)^t$ *where* $t = val_{\mathscr{C}_{r-1}}(i)$*; consequently,* $\mathscr{C}_{r-1}\langle i\rangle$ *is a prefix of* $\mathscr{C}_r\langle i\rangle$*, while* $\mathscr{C}_r\langle i\rangle$ *cannot be a prefix of* $\mathscr{C}_{r-1}\langle i\rangle$*.*

(iii) *If* $|\mathscr{C}_{r+1}\langle i\rangle| > 1$*, then there exist* $\xi \le r$ *and* $1 \le \rho \le m_\xi$*, so that* $\mathscr{C}_{r+1}\langle i\rangle = u_{(\xi,\, \rho)} \big(u_{(\xi,\, \xi)}\big)^t$ *where* $t = val_{\mathscr{C}_\xi}(i + |u_{(\xi,\, \rho)}|)$*.*

Lemma 6 shows how the Monge property of the system of all occurrences of all the group contexts propagates through a group configuration arrangement.

Lemma 6. *Referring to* $(\mathcal{B})$*, then for any* $1 \le r \le \hat{r}$*, the system* $\{\, \mathscr{C}_r\langle i\rangle \mid 1 \le i \le n \,\}$ *has the Monge property.*

Proof. We are going to prove it by induction over r. First consider the case when $r = 1$.
Then each $u_{(1,\, \ell)} = c$ for some letter c of x. Thus each $\mathscr{C}_1\langle i\rangle = c$ for some letter c of x, and hence either $\mathscr{C}_1\langle i\rangle = \mathscr{C}_1\langle j\rangle$ or $\mathscr{C}_1\langle i\rangle \cap \mathscr{C}_1\langle j\rangle = \emptyset$ for any $i \ne j$.
Induction hypothesis: $\{\, \mathscr{C}_{r-1}\langle i\rangle \mid 1 \le i \le n \,\}$ has the Monge property.
Consider $\mathscr{C}_r\langle i\rangle$ and $\mathscr{C}_r\langle j\rangle$ for $i \ne j$. WLOG assume $i < j$.

- Case $\mathscr{C}_r\langle i\rangle = \mathscr{C}_{r-1}\langle i\rangle$ and $\mathscr{C}_r\langle j\rangle = \mathscr{C}_{r-1}\langle j\rangle$.
  By the induction hypothesis, either $\mathscr{C}_{r-1}\langle i\rangle \cap \mathscr{C}_{r-1}\langle j\rangle = \emptyset$ or $\mathscr{C}_{r-1}\langle i\rangle$ includes $\mathscr{C}_{r-1}\langle j\rangle$.
- Case $\mathscr{C}_r\langle i\rangle = \mathscr{C}_{r-1}\langle i\rangle \big(\mathscr{C}_{r-1}\langle \ell\rangle\big)^\rho$, $prev_{\mathscr{C}_{r-1}}(\ell) = i$, $\rho = val_{\mathscr{C}_{r-1}}(\ell)$, $\mathscr{C}_{r-1}\langle \ell\rangle = u_{(r-1,\, r-1)}$, and $\mathscr{C}_r\langle j\rangle = \mathscr{C}_{r-1}\langle j\rangle$. Since for all involved components, no two can intersect, the possible subcases are:
  - $\mathscr{C}_{r-1}\langle j\rangle$ is disjoint from $\mathscr{C}_{r-1}\langle i\rangle \big(\mathscr{C}_{r-1}\langle \ell\rangle\big)^\rho$.
  - $\mathscr{C}_{r-1}\langle j\rangle$ is included in a copy of $\mathscr{C}_{r-1}\langle \ell\rangle$.
  - $\mathscr{C}_{r-1}\langle j\rangle$ is included in $\mathscr{C}_{r-1}\langle i\rangle$.
- Case $\mathscr{C}_r\langle i\rangle = \mathscr{C}_{r-1}\langle i\rangle$ and $\mathscr{C}_r\langle j\rangle = \mathscr{C}_{r-1}\langle j\rangle \big(\mathscr{C}_{r-1}\langle \ell\rangle\big)^\rho$, $prev_{\mathscr{C}_{r-1}}(\ell) = j$, $\rho = val_{\mathscr{C}_{r-1}}(\ell)$, and $\mathscr{C}_{r-1}\langle \ell\rangle = u_{(r-1,\, r-1)}$. Since for all involved components, no two can intersect, the possible subcases are:
  - $\mathscr{C}_{r-1}\langle i\rangle$ and $\mathscr{C}_{r-1}\langle j\rangle$ are disjoint.
  - $\mathscr{C}_{r-1}\langle i\rangle$ includes $\mathscr{C}_{r-1}\langle j\rangle$ as a proper suffix. By Obs. 5(i), it means that $\mathscr{C}_{r-1}\langle j\rangle = u_{(r-2,\, r-2)}$. Since $prev_{\mathscr{C}_{r-1}}(\ell) = j$, it means $u_{(r-2,\, r-2)} \prec u_{(r-1,\, r-1)}$, which is a contradiction as $u_{(r-1,\, r-1)} \prec u_{(r-1,\, r-2)} = u_{(r-2,\, r-2)}$.

- $\mathscr{C}_{r-1}\langle i\rangle$ includes $\mathscr{C}_{r-1}\langle j\rangle\big(\mathscr{C}_{r-1}\langle \ell\rangle\big)^{\xi}$ for some $\xi \le \rho$ as a proper suffix. Then $\mathscr{C}_{r-1}\langle \ell\rangle$ is a suffix of $\mathscr{C}_{r-1}\langle i\rangle$ and by $(\mathcal{B})$, $\mathscr{C}_{r-1}\langle \ell\rangle = u_{(r-2,\,r-2)}$, i.e. $u_{(r-1,\,r-1)} = u_{(r-2,\,r-2)}$, a contradiction.
- $\mathscr{C}_{r-1}\langle i\rangle$ includes $\mathscr{C}_{r-1}\langle j\rangle\big(\mathscr{C}_{r-1}\langle \ell\rangle\big)^{\rho}$ but not as a suffix.

· Case $\mathscr{C}_r\langle i\rangle = \mathscr{C}_{r-1}\langle i\rangle\big(\mathscr{C}_{r-1}\langle \ell\rangle\big)^{\rho}$, $prev_{\mathscr{C}_{r-1}}(\ell) = i$, $\rho = val_{\mathscr{C}_{r-1}}(\ell)$, $\mathscr{C}_{r-1}\langle \ell\rangle = u_{(r-1,\,r-1)}$, and $\mathscr{C}_r\langle j\rangle = \mathscr{C}_{r-1}\langle j\rangle\big(\mathscr{C}_{r-1}\langle k\rangle\big)^{\xi}$, $prev_{\mathscr{C}_{r-1}}(k) = j$, $\xi = val_{\mathscr{C}_{r-1}}(k)$, $\mathscr{C}_{r-1}\langle k\rangle = u_{(r-1,\,r-1)}$. Since for all involved components, no two can intersect, the possible subcases are:

- $\mathscr{C}_{r-1}\langle i\rangle\big(u_{(r-1,\,r-1)}\big)^{\rho}$ is disjoint from $\mathscr{C}_{r-1}\langle j\rangle\big(u_{(r-1,\,r-1)}\big)^{\xi}$.
- $\mathscr{C}_{r-1}\langle i\rangle$ includes $\mathscr{C}_{r-1}\langle j\rangle\big(u_{(r-1,\,r-1)}\big)^{\xi}$ but not as a suffix, which is fine.
- $\mathscr{C}_{r-1}\langle i\rangle$ includes $\mathscr{C}_{r-1}\langle j\rangle\big(u_{(r-1,\,r-1)}\big)^{\xi}$ as a suffix, which is a contradiction as the ξ copies of $u_{(r-1,\,r-1)}$ are immediately followed by another ρ copies, so $\mathscr{C}_r\langle j\rangle$ should equal to $\mathscr{C}_{r-1}\langle j\rangle\big(\mathscr{C}_{r-1}\langle k\rangle\big)^{\xi+\rho}$.
- $\mathscr{C}_{r-1}\langle i\rangle$ includes $\mathscr{C}_{r-1}\langle j\rangle\big(u_{(r-1,\,r-1)}\big)^{\tau}$, $\tau < \xi$ as a proper suffix. But then $\xi - \tau = \rho$ and $\ell = k + \tau|u_{(r-1,\,r-1)}|$. Since $prev_{\mathscr{C}_{r-1}}(\ell) = i$ since $\mathscr{C}_r\langle i\rangle = \mathscr{C}_{r-1}\langle i\rangle\big(u_{(r-1,\,r-1)}\big)^{\rho}$ and $prev_{\mathscr{C}_{r-1}}(\ell) = j$ since $\mathscr{C}_r\langle j\rangle = \mathscr{C}_{r-1}\langle j\rangle\big(u_{(r-1,\,r-1)}\big)^{\xi}$, a contradiction.
- $\mathscr{C}_{r-1}\langle j\rangle$ is a suffix of $\mathscr{C}_{r-1}\langle i\rangle$ and $\rho = \xi$. Then $\ell = k$ and $prev_{\mathscr{C}_{r-1}}(\ell) = i$ since $\mathscr{C}_r\langle i\rangle = \mathscr{C}_{r-1}\langle j\rangle\big(u_{(r-1,\,r-1)}\big)^{\rho}$ and $prev_{\mathscr{C}_{r-1}}(\ell) = j$ since $\mathscr{C}_r\langle j\rangle = \mathscr{C}_{r-1}\langle j\rangle\big(u_{(r-1,\,r-1)}\big)^{\xi}$, a contradiction.

□

The process of refinement of the groups has a very particular property, namely $u_{(r,\,k)}$ is never followed immediately by $u_{(r-\xi,\,r-\xi)}$ for any $r-\xi \ge 1$ if $u_{(r,\,k)} \prec u_{(r-\xi,\,r-\xi)}$.

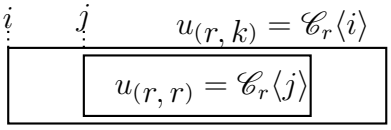
Lemma 7. *Referring to* $(\mathcal{B})$*, let* $1 \le r-\xi < r \le \hat{r}$*, let* $u_{(r,\,k)} \prec u_{(r-\xi,\,r-\xi)}$*, and let* $j = i+|u_{(r,\,k)}|$*. Then it is impossible to have* $\mathscr{C}_r\langle i\rangle = u_{(r,\,k)}$ *and* $\mathscr{C}_r\langle j\rangle = u_{(r-\xi,\,r-\xi)}$.

Proof. Arguing by contradiction assume to have it for some i, j, ξ, r, and k. Then $\mathscr{C}_{r-\xi}\langle i\rangle \trianglelefteq \mathscr{C}_r\langle i\rangle = u_{(r,\,k)} \prec u_{(r,\,r-\xi)} = u_{(r-\xi,\,r-\xi)}$, and so $prev_{\mathscr{C}_{r-\xi}}(j) \ge i$. If $prev_{\mathscr{C}_{r-\xi}}(j) = i$, then $\mathscr{C}_r\langle i\rangle \triangleleft \mathscr{C}_{r-\xi+1}\langle i\rangle \trianglelefteq \mathscr{C}_r\langle i\rangle$, a contradiction. Thus, $j_1 = prev_{\mathscr{C}_{r-\xi}}(j) > i$. So, $\mathscr{C}_{r-\xi+1}\langle j_1\rangle$ and $\mathscr{C}_r\langle i\rangle$ intersect, and so $\mathscr{C}_r\langle j_1\rangle$ and $\mathscr{C}_r\langle i\rangle$ intersect as $\mathscr{C}_{r-\xi+1}\langle j_1\rangle \trianglelefteq \mathscr{C}_r\langle j_1\rangle$. But that contradicts Lemma 6. □

We need to ascertain that the definition of the processing of $G_{u_{(r,\,r)}}$ can be carried out as defined in Def. 4, i.e. that the property of *prev* propagates through a group configurations arrangement. Lemma 8 shows that.

Lemma 8. *Referring to* ($\mathcal{B}$)*, for any* $1 \le r \le \hat{r}$*, for any* $i \in G_{u_{(r,\,r)}}$*, if* $j = prev_{\mathscr{C}_r}(i)$*, then* $x[j\,..\,n]$ *has* $\mathscr{C}_r\langle j\rangle(u_{(r,\,r)})^t$ *where* $t = val_{\mathscr{C}_r}(i)$*, as a prefix.*

Proof. It is clear that for $r = 1$ it is true: Let c be the largest letter in x, then $G_{u_{(1,\,1)}} = G_c$. Let $i \in G_c$ and let $j = prev_{\mathscr{C}_1}(i)$. Let $i_1 = \min\{\ell \le i \mid x[\ell] = c\}$, then $val_{\mathscr{C}_1}(i) = val_{\mathscr{C}_1}(i_1) = t$ and $prev_{\mathscr{C}_1}(i_1) = prev_{\mathscr{C}_1}(i) = j$. Then $j = i_1{-}1$, $\mathscr{C}_1\langle j\rangle = b$ for some $b \prec c$, and $\mathscr{C}_1\langle i_1\rangle = c^t$.
We are assuming it holds true for r.
We shall prove it for $r{+}1$, but first we need to prove three simple claims.
Though a bit stronger, in essence, the first claim states that the configuration illustrated below cannot happen.



Claim 1: *If* $1 \le r \le \hat{r}$*,* $i, j \in 1\,..\,n$*,* $i < j$*, and* $\xi \ge 0$*, then* $\mathscr{C}_r\langle i\rangle$ *cannot include* $\mathscr{C}_{r+\xi}\langle j\rangle = u_{(r+\xi,\,r+\xi)}$.

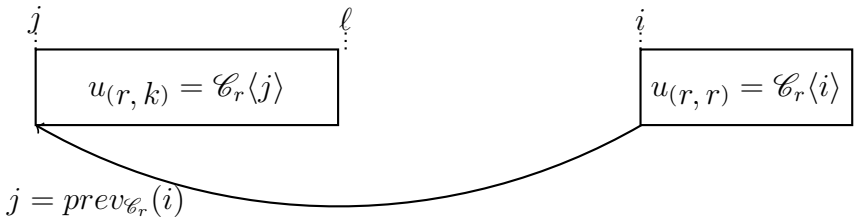
Arguing by contradiction, take the minimal r such that for some $i < j$, and some ξ, $\mathscr{C}_r\langle i\rangle$ includes $\mathscr{C}_{r+\xi}\langle j\rangle = u_{(r+\xi,\,r+\xi)}$. Let $\mathscr{C}_r\langle i\rangle = u_{(r,\,k)}$ for some k. Since $\mathscr{C}_r\langle i\rangle$ includes $\mathscr{C}_{r+\xi}\langle j\rangle$ and $\mathscr{C}_r\langle i\rangle \ne \mathscr{C}_{r+\xi}\langle j\rangle$ as $i < j$, $|\mathscr{C}_r\langle i\rangle| \ge 2$ and so $r > 1$. By Obs. 5(iii), there are $r' < r$, k', and $t \ge 1$ so that $u_{(r,\,k)} = \mathscr{C}_r\langle i\rangle = u_{(r',\,k')}\big(u_{(r',\,r')}\big)^t$. For $0 \le h < t$, define $\ell_h = i{+}|u_{(r',\,k')}| + h|u_{(r',\,r')}|$. Then $\mathscr{C}_{r'}\langle i\rangle = u_{(r',\,k')}$ and each $\mathscr{C}_{r'}\langle \ell_h\rangle = u_{(r',\,r')}$. For any $0 \le h < t$, by Lemma 6, $u_{(r+\xi,\,r+\xi)} = \mathscr{C}_{r+\xi}\langle j\rangle$ must be either disjoint from $u_{(r',\,r')} = u_{(r+\xi,\,r')} = \mathscr{C}_{r+\xi}\langle \ell_h\rangle$, or one must include the other. Let $\xi' = r{-}r'{+}\xi$, then $\xi' \ge 0$ and $r{+}\xi = r'{+}\xi'$.

- If $\mathscr{C}_{r+\xi}\langle j\rangle$ is disjoint from every $\mathscr{C}_{r+\xi}\langle \ell_h\rangle$, then $u_{(r'+\xi',\,r'+\xi')} = u_{(r+\xi,\,r+\xi)}$ must be included in $u_{(r',\,k')} = \mathscr{C}_{r'}\langle i\rangle$. So $u_{(r'+\xi',\,r'+\xi')}$ is included in $u_{(r',\,k')} = \mathscr{C}_{r'}\langle i\rangle$, which contradicts the minamility of r.
- Thus, $\mathscr{C}_{r+\xi}\langle j\rangle$ must be included in or include $\mathscr{C}_{r+\xi}\langle \ell_h\rangle$ for some $h \in 0..t{-}1$. If $\mathscr{C}_{r+\xi}\langle \ell_h\rangle = u_{(r',\,r')}$ included $\mathscr{C}_{r+\xi}\langle j\rangle = u_{(r'+\xi',\,r'+\xi')}$, we would have a contradiction with the minimality of r. Thus $\mathscr{C}_{r+\xi}\langle j\rangle = u_{(r'+\xi',\,r'+\xi')}$ must include $\mathscr{C}_{r+\xi}\langle \ell_h\rangle = u_{(r',\,r')}$, and so $j = \ell_h$, $u_{(r'+\xi',\,r'+\xi')} = u_{(r',\,r')}$, and so $r' = r'{+}\xi'$, a contradiction.

This concludes the proof of Claim 1.

Claim 2: *Let* $j = prev_{\mathscr{C}_r}(i)$*,* $\mathscr{C}_r\langle i\rangle = u_{(r,\,r)}$*, and for any* $j < i' < i$*,* $\mathscr{C}_r\langle i'\rangle \ne u_{(r,\,r)}$*. Then* $\mathscr{C}_r\langle j\rangle$ *and* $\mathscr{C}_r\langle i\rangle$ *are adjacent.*

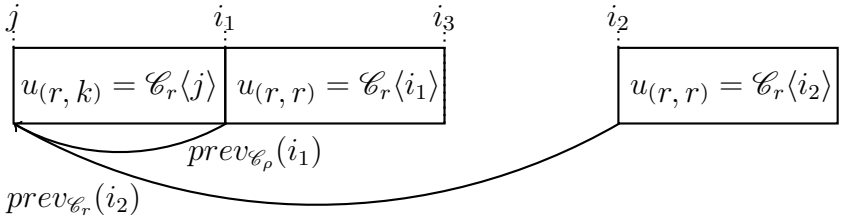
By Claim 1, $\mathscr{C}_r\langle j\rangle$ cannot include $\mathscr{C}_r\langle i\rangle$, and so, by Lemma 6, $\mathscr{C}_r\langle j\rangle$ and $\mathscr{C}_r\langle i\rangle$ are disjoint. By contradiction, we shall see that they must be adjacent. So assume that they are not adjacent,



i.e. $\mathscr{C}_r\langle \ell\rangle \succcurlyeq u_{(r,\,r)}$, therefore $\mathscr{C}_r\langle \ell\rangle = u_{(r,\,\rho)} = u_{(\rho,\,\rho)}$ for some $\rho \le r$. But since there is no occurrence of $u_{(r,\,r)}$ between j and i, $\rho < r$. But this is not possible by Lemma 7. This concludes the proof of Claim 2.

Claim 3: *Let $j = prev_{\mathscr{C}_r}(i_1) = prev_{\mathscr{C}_r}(i_2)$ and let $\mathscr{C}_r\langle i_1\rangle = \mathscr{C}_r\langle i_2\rangle = u_{(r,\,r)}$, and let $i_2 > i_1 + |u_{(r,\,r)}|$. Then $\mathscr{C}_r\langle i_1 + |u_{(r,\,r)}|\rangle = u_{(r,\,r)}$.*

Let $i_3 = i_1 + |u_{(r,\,r)}|$. Since $j = prev_{\mathscr{C}_r}(i_2)$, it follows that $i_3 \in \bigcup_{\xi=1}^{r} G_{u_{(r,\,\xi)}}$. If $i_3 \in \bigcup_{\xi=1}^{r-1} G_{u_{(r,\,\xi)}}$,



then $\mathscr{C}_r\langle i_3\rangle = u_{(r,\,\xi)} = u_{(\xi,\,\xi)}$ for some $\xi < r$. Since $\mathscr{C}_\xi\langle i_1\rangle \trianglelefteq \mathscr{C}_r\langle i_1\rangle \prec u_{(r,\,\xi)}$, it follows that $i_4 = prev_{\mathscr{C}_\xi}(i_3) \ge i_1$. If $prev_{\mathscr{C}_\xi}(i_3) = i_1$, then for some $t \ge 1$, $\mathscr{C}_{\xi+1}\langle i_1\rangle = u_{(r,\,r)}\big(u_{(\xi,\,\xi)}\big)^t \trianglelefteq \mathscr{C}_r\langle i_1\rangle = u_{(r,\,r)}$, a contradiction. Thus $i_4 > i_1$ and so $\mathscr{C}_{\xi+1}\langle i_4\rangle = x[i_4\,..\,i_3{-}1]\big(u_{(\xi,\,\xi)}\big)^t \trianglelefteq \mathscr{C}_r\langle i_4\rangle$ and so $\mathscr{C}_r\langle i_1\rangle$ and $\mathscr{C}_r\langle i_4\rangle$ intersect, a contradiction with Lemma 6. Thus, $i_3 \in G_{u_{(r,\,r)}}$.

Now we can prove the induction step. Let $j = prev_{\mathscr{C}_r}(i)$, let $p = |u_{(r,\,r)}|$, and let $\mathscr{C}_r\langle i\rangle = u_{(r,\,r)}$. Let ℓ be the smallest i such that $j = prev_{\mathscr{C}_r}(i)$ and $\mathscr{C}_r\langle i\rangle = u_{(r,\,r)}$. Then by Claim 3, $x[i_1\,..\,i_1{+}tp{-}1] = \big(u_{(r,\,r)}\big)^t$ where $t = val_{\mathscr{C}_r}(\ell)$, moreover $x[\ell{-}p\,..\,\ell{-}1] \ne u_{(r,\,r)}$ and $x[i{+}tp\,..\,i{+}(t{+}1)p{-}1] \ne u_{(r,\,r)}$. By Claim 2, $\mathscr{C}_r\langle j\rangle$ and $\mathscr{C}_r\langle \ell\rangle$ are adjacent, and so $x[j\,..\,n]$ has $\mathscr{C}_r\langle j\rangle\big(u_{(r,\,r)}\big)^t$ as a prefix. □

Note that for $r = 1$, $G_{u_{(1,\,1)}} = G_{a_k}$ where a_k is the largest letter of x, and so G_{a_k} is complete. Lemma 9 shows how the processing of $G_{u_{(r,\,r)}}$ makes the group $G_{u_{(r+1,\,r+1)}}$ complete, i.e. how it propagates through the refinement process.

Lemma 9. *Referring to* ($\mathcal{B}$)*, for any* $1 \leq r < \hat{r}$*, after processing the group* $G_{u_{(r,\,r)}}$*, the group* $G_{u_{(r+1,\,r+1)}}$ *is complete.*

Proof. Assume to have $i \in G_{u_{(r+1,\,r+1)}}$ so that $u_{(r+1,\,r+1)} = \mathscr{C}_{r+1}\langle i\rangle = \mathscr{C}_r\langle i\rangle = u_{(r,\,r+1)}$ is not maximal, i.e. $u_{(r+1,\,r+1)} \preccurlyeq x[j\,..\,n]$, where $j = i+|u_{(r+1,\,r+1)}|$. There are two cases.

- $G_{u_{(r+1,\,r+1)}} = G_{u_{(r,\,r+1)}}$.
  Then for some $t \geq 1$, and some ξ, $x[j\,..\,n]$ has $\big(u_{(r,\,r+1)}\big)^t u_{(r,\,\xi)}$ as a prefix and $u_{(r,\,\xi)}$ is not a prefix of $u_{(r,\,r+1)}$, and $\mathscr{C}_{r+1}\langle i\rangle = u_{(r+1,\,r+1)} = u_{(r,\,r+1)} \preccurlyeq \big(u_{(r,\,r+1)}\big)^t u_{(r,\,\xi)}$, and so $u_{(r,\,r+1)} \prec\!\cdot\, u_{(r,\,\xi)}$, and so $u_{(r,\,r)} \preccurlyeq u_{(r,\,\xi)}$ and $\xi \leq r$. Then we have $u_{(r+1,\,r+1)}$ immediately followed by $u_{(\xi,\,\xi)}$, $\xi < r{+}1$, and $u_{(r+1,\,r+1)} \prec u_{(\xi,\,\xi)}$, which contradicts Lemma 7.
- $G_{u_{(r+1,\,r+1)}}$ was created from $G_{u_{(r,\,r+1)}}$ and so $\mathscr{C}_{r+1}\langle i\rangle = u_{(r,\,r+1)}\big(u_{(r,\,r)}\big)^t$ for $t = val_{\mathscr{C}_r}(j)$. Let $\ell = i{+}t|u_{(r,\,r+1)}|$. There are three subcases:

  (α) For some $p \geq 1$, some $t_1 > t$, $x[j\,..\,n]$ has as a prefix $\Big(u_{(r,\,r+1)}\big(u_{(r,\,r)}\big)^t\Big)^p u_{(r,\,r+1)}\big(u_{(r,\,r)}\big)^{t_1}$. But then there is a group that has $u_{(r,\,r+1)}\big(u_{(r,\,r)}\big)^{t_2}$, $t_2 \geq t_1$, as the context, which means that the group with the context $u_{(r,\,r+1)}\big(u_{(r,\,r)}\big)^t$ cannot be $G_{u_{(r+1,\,r+1)}}$, a contradiction.

  (β) For some $p \geq 1$, some $t_1 < t$, $x[j\,..\,n]$ has as a prefix $\Big(u_{(r,\,r+1)}\big(u_{(r,\,r)}\big)^t\Big)^p u_{(r,\,r+1)}\big(u_{(r,\,r)}\big)^{t_1} u_{(r,\,\xi)}$ and $u_{(r,\,\xi)} \neq u_{(r,\,r)}$. Since $u_{(r,\,r+1)} \prec u_{(r,\,\xi)}$, it follows that $\xi \leq r$, but since $u_{(r,\,\xi)} \neq u_{(r,\,r)}$, we have $\xi < r$. So $u_{(r,\,\xi)} = u_{(\xi,\,\xi)}$. But then we have $u_{(r,\,r)}$ immediately followed by $u_{(\xi,\,\xi)}$ with $u_{(r,\,r)} \prec u_{(\xi,\,\xi)}$, which contradicts Lemma 7.

  (γ) For some $p \geq 1$ and some ξ, $x[j\,..\,n]$ has $\Big(u_{(r,\,r+1)}\big(u_{(r,\,r)}\big)^t\Big)^p u_{(r,\,\xi)}$ as a prefix and so that $u_{(r,\,r+1)} \prec\!\cdot\, u_{(r,\,\xi)}$. It follows that $\xi \leq r$. If $u_{(r,\,\xi)} = u_{(r,\,r)}$, then this would be case (α) as $x[j\,..\,n]$ would have $\Big(u_{(r,\,r+1)}\big(u_{(r,\,r)}\big)^t\Big)^{p-1} u_{(r,\,r+1)}\big(u_{(r,\,r)}\big)^{t+1}$ as a prefix. Thus we can assume that $\xi < r$ and so $u_{(r,\,\xi)} = u_{(\xi,\,\xi)}$. But then we have $u_{(r,\,r)}$ immediately followed by $u_{(\xi,\,\xi)}$ with $u_{(r,\,r)} \prec u_{(\xi,\,\xi)}$, which contradicts Lemma 7.

Thus, we showed that for any $i \in G_{u_{(r+1,\,r+1)}}$, $\mathscr{C}_{r+1}\langle i\rangle$ is a maximal Lyndon factor, i.e. $G_{u_{(r+1,\,r+1)}}$ is complete. □

Theorem 10. *Referring to* $(\mathcal{B})$, $\mathscr{C}_{\hat{r}} = \left[G_{u_{(}\hat{r}, m_{\hat{r}})}, G_{u_{(}\hat{r}, m_{\hat{r}}-1)} \ldots, G_{u_{(}\hat{r}, 1)}\right]$ *is an* $\hat{r}$*-proper configuration.*

Proof. It is straightforward to see that $\mathscr{C}_{\hat{r}}$ satisfies Def. 2(i)-(v). That it satisfies Def. 2(vi) follows from Lemma 6. Thus, $\mathscr{C}_{\hat{r}}$ is a group configuration. That Def. 4(vii) is satisfied follows from Lemma 9. That Def. 4(viii) is satisfied follows from the fact that $\mathscr{C}_{\hat{r}}$ was obtained by processing of $G_{u_{(}\hat{r}-1, \hat{r}-1)}$. Finally, the fact that Def. 4(ix) is satisfied follows from Lemma 8. □

Theorem 11. *For a string* x*, Baier's sort identifies and sorts all maximal Lyndon factors of* x*.*

Proof. Consider $(\mathcal{B})$ when the process of refinement stops. Consider an index i. There is a unique $1 \leq \ell \leq m_{\hat{r}}$ so that $i \in G_{u_{(}\hat{r}, \ell)}$. Thus $x[i\,..\,n]$ has $u_{(}\hat{r}, \hat{r})$ as a prefix. Since $G_{u_{(}\hat{r}, \ell)}$ is complete, $x[i\,..\,i+|u_{(}\hat{r}, \ell)|-1] = u_{(}\hat{r}, \ell)$ is a maximal Lyndon factor. Hence all maximal Lyndon factors of x are accounted for. □

5 Conclusion and future work

We showed how the process of refinement of the initial configuration propagates and is sustained as long as there are some i's in the group being processed such that $prev(i) \neq \texttt{nil}$. We showed that this process will identify and present in a sorted way all maximal Lyndon factors of a given string.

An algorithmic analysis of the process and implementation details are a necessary further step. Baier's implementation of phase I of his algorithm is linear in time, due to the use of the *prev* operator. The *prev*() values can be computed for the initial configuration in $\mathcal{O}(n)$ steps using a stack of size n, where n is the length of the input string, and then during the processing they can be updated in $\mathcal{O}(k)$ steps to work properly for the next level of refinement, where k is the the size of the group being processed. Our introduction of the *valence* makes the treatment of repetitions of the context of the group being processed more mathematically sound and straightforward and it lends itself to a simpler algorithmic treatment allowing to process the indices of the group being processed simply from the largest to the smallest with very little overhead. Our `C++` implementation uses all these mathematical insights and thus only uses memory of $13n$ integers, a significant improvement over Baier's implementation. Moreover, our implementation uses a static approach, so all required memory of $13n$ integers is allocated once and no further dynamic memory allocation is required, significantly speeding up the execution. Of course, rigorous comparison testing of the performances of the two implementations is needed before any conclusion can be drawn. The code can be obtained from

`http://www.cas.mcmaster.ca/~franek/research/bls.cpp`

References

1. U. Baier: *Linear-time suffix sorting — a new approach for suffix array construction.* M.Sc. Thesis, University of Ulm, 2015.
2. U. Baier: *Linear-time suffix sorting — a new approach for suffix array construction*, in 27th Annual Symposium on Combinatorial Pattern Matching (CPM 2016), R. Grossi and M. Lewenstein, eds., vol. 54 of Leibniz International Proceedings in Informatics (LIPIcs), Dagstuhl, Germany, 2016, Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, pp. 23:1–23:12.
3. H. Bannai, T. I, S. Inenaga, Y. Nakashima, M. Takeda, and K. Tsuruta: *The "Runs" Theorem.* SIAM J. COMPUT., 46 2017, pp. 1501–1514.
4. G. Chen, S. Puglisi, and W. Smyth: *Lempel-Ziv factorization using less time and space*, in Mathematics in Computer Science 1-4, J. Chan and M. Crochemore, eds., 2008, pp. 482–488.
5. K. T. Chen, R. H. Fox, and R. C. Lyndon: *Free differential calculus. iv. the quotient groups of the lower central series.* Annals of Mathematics, 68(1) 1958, pp. 81–95.
6. M. Crochemore, L. Ilie, and W. Smyth: *A simple algorithm for computing the Lempel-Ziv factorization*, in Proc. 18th Data Compression Conference, J. Storer and M. Marcellin, eds., 2008, pp. 482–488.
7. J. Daykin, F. Franek, J. Holub, A. S. Islam, and W. Smyth: *Reconstructing a string from its Lyndon arrays.* Theoretical Computer Science, 710 2018, pp. 44–51.
8. J.-P. Duval: *Factorizing words over an ordered alphabet.* J. Algorithms, 4(4) 1983, pp. 363–381.
9. F. Franek, A. S. Islam, M. S. Rahman, and W. Smyth: *Algorithms to compute the Lyndon array*, in Proceedings of Prague Stringology Conference 2016, PSC'16, 2016, pp. 172–184.
10. F. Franek, A. Paracha, and W. Smyth: *The linear equivalence of the suffix array and the partially sorted Lyndon array*, in Proc. Prague Stringology Conference, 2017, pp. 77–84.
11. A. Glen, J. Simpson, and W. Smyth: *Counting Lyndon factors.* Electronic J. Combinatorics, 24 2017, pp. 3–28.
12. C. Hohlweg and C. Reutenauer: *Lyndon words, permutations and trees.* Theoretical Computer Science, 30(1) 2003, pp. 173–178.
13. M. Lothaire: *Combinatorics on words*, Addison-Wesley, Reading, Mass., 1983.
14. F. Louza, G. Manzini, W. Smyth, and G. Telles: *Lyndon array construction during Burrows-Wheeler inversion.* submitted for publication, 2017.
15. G. Melancon: *Lyndon word*, in Encyclopedia of Mathematics, M. Hazewinkel, ed., Springer Science+Business Media B.V. / Kluwer Academic Publishers, 2001.
16. J. Sawada and F. Ruskey: *Generating Lyndon brackets.* Journal of Algorithms, 46 2003, pp. 21–26.